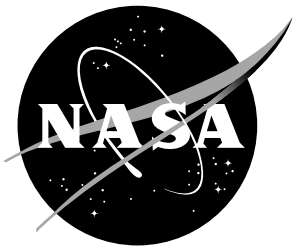
NASA/TM-2020-5006086



LaRC SmartLab Apps For Instrument Control and Data Processing: Laboratory Environment Monitor

Godfrey Sauti
NASA Langley Research Center, Hampton, Virginia

Benjamin D. Jensen
NASA Langley Research Center, Hampton, Virginia

James D. Tobin
Christopher Newport University, Newport News, Virginia

Mathew L. Bartgis
Christopher Newport University, Newport News, Virginia

Emilie J. Siochi
NASA Langley Research Center, Hampton, Virginia

August 2020

NASA STI Program. . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collection of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

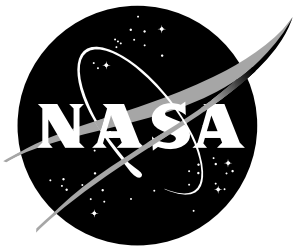
- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.
- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.
- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.
- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI Program, see the following:

- Access the NASA STI program home page at `http://www.sti.nasa.gov`
- E-mail your question to `help@sti.nasa.gov`
- Phone the NASA STI Information Desk at 757-864-9658
- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM-2020-5006086



LaRC SmartLab Apps For Instrument Control and Data Processing: Laboratory Environment Monitor

Godfrey Sauti
NASA Langley Research Center, Hampton, Virginia

Benjamin D. Jensen
NASA Langley Research Center, Hampton, Virginia

James D. Tobin
Christopher Newport University, Newport News, Virginia

Mathew L. Bartgis
Christopher Newport University, Newport News, Virginia

Emilie J. Siochi
NASA Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

August 2020

Acknowledgments

Quinton Duncan, Keith Gordon and Scott Zavada for assistance with the location and installation of test probes.

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Abstract

The **LaRC SmartLab** applications are a series of software tools to greatly enhance researcher efficiency by streamlining and automating workflows. Python scripts and applications are increasingly being used in scientific workflows, including for instrument control and data processing. Interactive Python scripting environments such as JupyterLab provide powerful tools for using Python. In some use cases, the development of standalone applications with dedicated graphical user interfaces can enhance the utility of the code and open it up to more users, including non-programmers. Here, we describe a Python based application for communicating with, and displaying data from, iTHX Temperature, Humidity, and Dew Point probes. We discuss the set up and use of the application as well as its implementation. We also highlight the use of `Simulated` probes to enable users and developers to familiarize with or debug the application, even when they do not have access to the physical hardware in the laboratory.

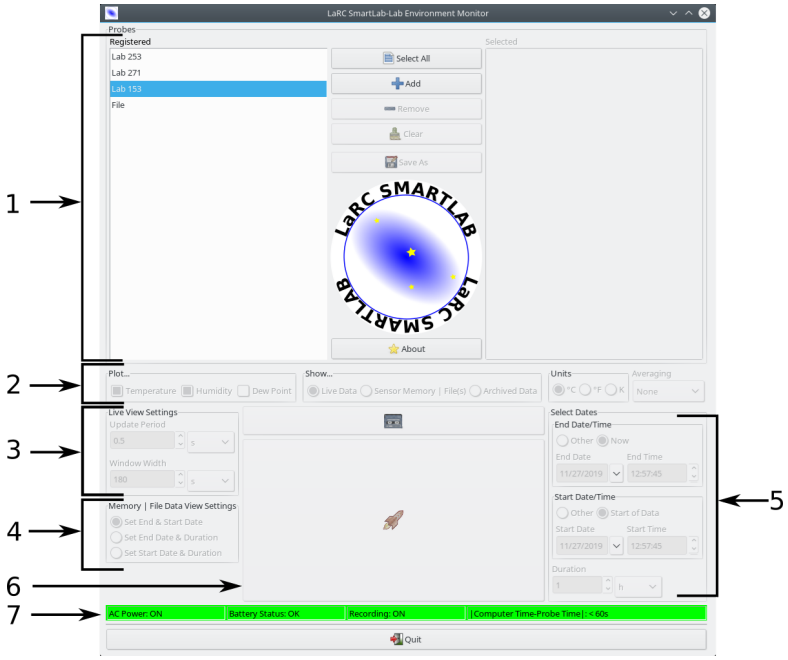
THIS PAGE INTENTIONALLY LEFT BLANK

Contents

lsltemp Quick Start Guide

LaRC SmartLab Team*

August 4, 2020



1. Select probe(s) or file(s) whose data to display
2. Select the environmental data to be displayed, source of the data and the units
3. Set the display time window for plotting live data from the probes
4. Select how to set date/time ranges when plotting data from probe memory or files
5. Select the date range for the display of data from probe memory or files
6. Plot data from sensors that are online, simulated probes or files using the current settings
7. See the status of the currently highlighted probe

*POC: godfrey.sauti-1@nasa.gov

1 Introduction

The **LaRC SmartLab**  project aims to develop tools to enhance the productivity of the researcher. Communication with instruments is a critical part of the research workflow and therefore instrument control applications are among the tools being developed by the project. Automating the data gathering process can provide the researcher with valuable information while minimizing the time and effort dedicated to collecting that data. Here, we discuss an application that continuously measures and communicates the laboratory Temperature, Humidity, and Dew Point. This application is developed in Python[1] utilizing the wxPython framework[2] to provide a graphical user interface (GUI). The GUI enables the application to be easily usable by both Python programmers and non-programmers alike. The application uses Python networking tools to interact with the temperature and humidity probes over a wired and wireless Local-Area Network (LAN).

We start by describing the hardware that interfaces with the application in Section 2. The controls that are available in the application GUI and their functions are described in Section 3. In Section 4 we discuss how to get the application running and the information that the user receives upon startup. We present detailed use cases of the application, with examples, in Section 5. We will highlight the use of `Simulated` probes, a feature of the application that enables users and developers to familiarize with or debug it, even when they do not have access to the hardware in the laboratory (Section 5.3). A brief description of key aspects of the implementation of the application code is provided in Section 6. This description is not meant as a complete listing of the application source code, which can be obtained from the authors.

The application is available not only as Python source code but also as a Microsoft Windows installer. In Appendix A we show some examples from an installed instance of the application running on MS Windows 10.

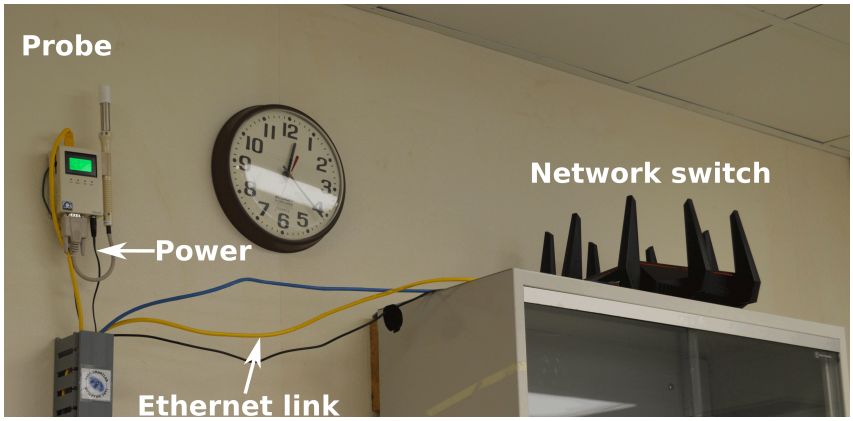
In addition to this document, there are videos with examples of the usage of the application, also available from the authors. We briefly list the content of these videos in Appendix B.

2 Laboratory Environment Monitoring

Constant monitoring of the laboratory temperature and humidity can provide data that can be correlated to experiments. In addition to any notes that the researcher takes down at the start or end of an experiment, or the data provided by test instruments with temperature control/monitoring, a constant stream of the laboratory conditions can be useful in figuring out what happened at any point before, during, and after an experiment. Conditions, such as what a sample or chemicals sitting on a laboratory bench might have been exposed to, prior to a measurement or synthesis experiment, may have a bearing on the interpretation of the results. The hardware and software application described here provide that constant stream of data and include features that enable capture and storage of data even in cases of power outages.

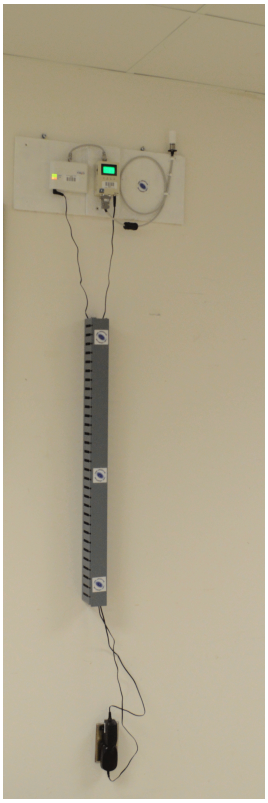
2.1 Hardware used

The **lsltemp** application interfaces with Omega iTHX-SD Chart Recorders for Temperature & Humidity (Omega Engineering Inc., Stamford CT) to monitor the temperature, humidity and dew point. These probes have Ethernet connectivity, a battery for backup power and an SD card for on-board data storage. While each iTHX probe features an on-board web-server for configuration and data access and multiple probes can be accessed through a JAVA™ Applet[3], the **lsltemp** application provides access using Python which is becoming a widely used language in the research community. For the in-laboratory tests of **lsltemp**, connectivity between the probes and the computers running the application was provided by an HP Procurve J4897 24-Port Fast Ethernet Gigabit Rj45 Unmanaged Network Switch. For probes that could not be directly connected via Ethernet, an ASUS RT-AC5300 Tri-band WiFi Router (ASUSTek Computer Inc.) and Silex SX-BR-4600WAN2 Gigabit Ethernet to 802.11a/b/g/n Wireless Bridge (Silex Technology, Santa Ana, CA) were used. The router and bridge were configured to communicate using their 5 GHz radios with their 2.4 GHz radios disabled. The probes, at locations in three separate laboratories, are shown in Figure 1.



(a) Lab A

(b) Lab B



(c) Lab C

Figure 1: Temperature and humidity probes for laboratory environment monitoring. The in-laboratory application test setup consisted of probes in three laboratories connected to computers running **lsltemp** via an Ethernet and wireless network.

3 Graphical User Interface

In this section, we describe the application's GUI which consists of several controls and a status bar. We also discuss modal dialog boxes that are generated at run-time to communicate specific information to the user.

Figure 2 shows the application GUI as it appears on startup with most of the controls disabled (see Section 3.2).

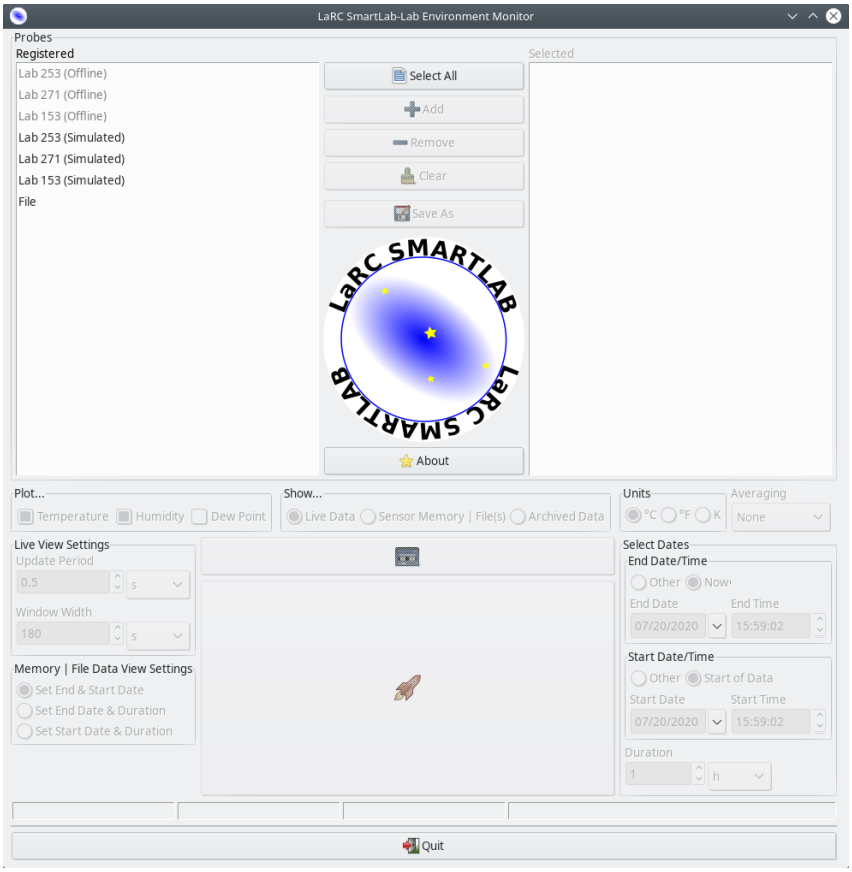


Figure 2: The GUI of the application as it appears on startup.

3.1 The controls

Figure 3 shows the GUI with the controls/control groups labeled. The functions of the controls/control groups are described below.

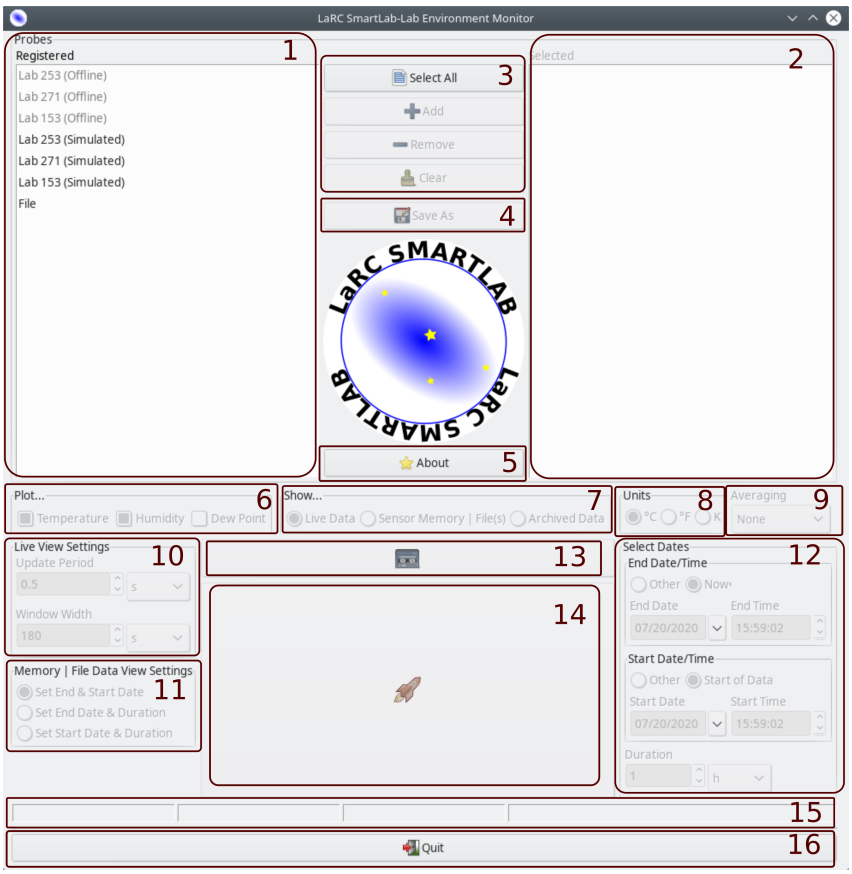


Figure 3: Controls of the lab environment monitoring application with numbering for ease of description.

1. **Registered (probes):** Identities of the probes read in from the application's configuration file. The physical probes may be online or offline. If all the physical probes are offline, the application will generate a set of `Simulated` probes. These simulated probes enable the user to interact with the rest of the functionality of the application as they would with physical probes online. The `Registered` probes selector also includes a special entry for selecting files. This is used to pick previously recorded and downloaded data for display.

2. **Selected (probes):** The probes (and files) selected to have their data displayed.

3. **Probe selection buttons:** Used to select and deselect the probes and files whose data to display

4. **Save As button:** Allows the user to save the currently highlighted probe's data to a file

5. **About button:** Displays information about the application

6. **Plot selectors:** For choosing which combination of the Temperature, Humidity, and Dew Point is to be displayed

7. **Data source selectors:** For choosing the source of the data to be displayed

8. **Units selectors:** For picking the units in which data are to be displayed

9. **Averaging:** Enables time averaging to be applied to data from sensor memory or files before it is displayed

10. **Live view settings:** Time settings for the display of live streamed sensor data

11. **Memory | File data settings:** Options for how to set the date range used for displaying data from sensor memory or files

12. **Dates selectors:** Date ranges for the display of data from sensor memory or files

13. **Record button:** (Re)starting of data recording to a highlighted probe's on-board memory

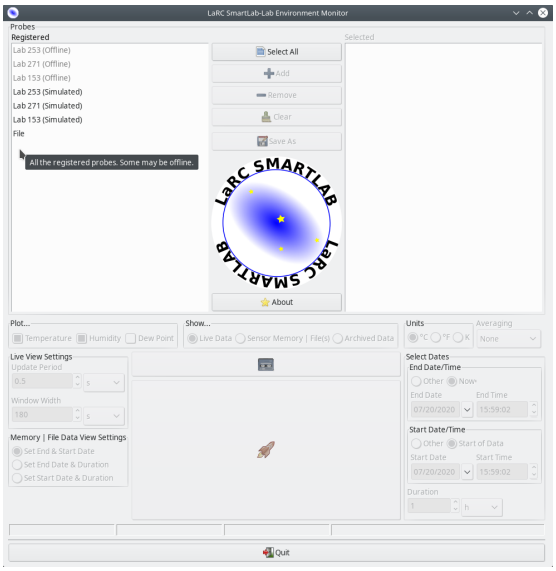
14. **Execute button:** Starts the display of data from the selected probes and or files

15. **Status bar:** Displays messages about the status of the probes

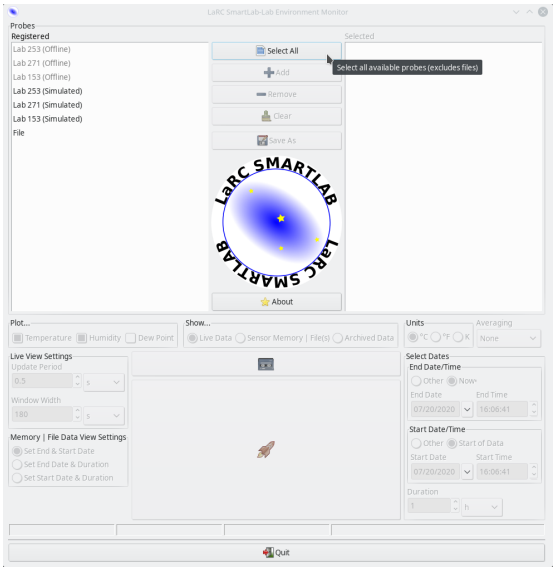
16. **Quit button:** Exits the application

3.2 Guiding the user

The application includes various features to guide the user. These features include the enabling and disabling of controls, depending on the probe status, data loaded, and display settings. Another feature for guiding the user are control tooltips. Each control comes with this short description of its function which is displayed when the user's mouse is hovering over it (Figure 4).



(a) Tooltip for the `Registered` probes column



(b) Tooltip for the Select All probes button

Figure 4: Examples of the control tooltips that provide the user with information about their functions.

3.3 The status bar

Once the application starts up (more details in Section 4), the user is able to highlight any one of the probes that are enabled for selection. When a probe that is online is highlighted, **lsltemp** displays information about the status of that probe as shown in Figures 5 and 6. Figure 5 shows the status bar messages when the probe parameters that are checked by the application are in their accepted ranges. Figures 6(a) and 6(b) show the status bar with the messages that are displayed when one or more of the highlighted probe's tracked parameters are outside of the accepted range.

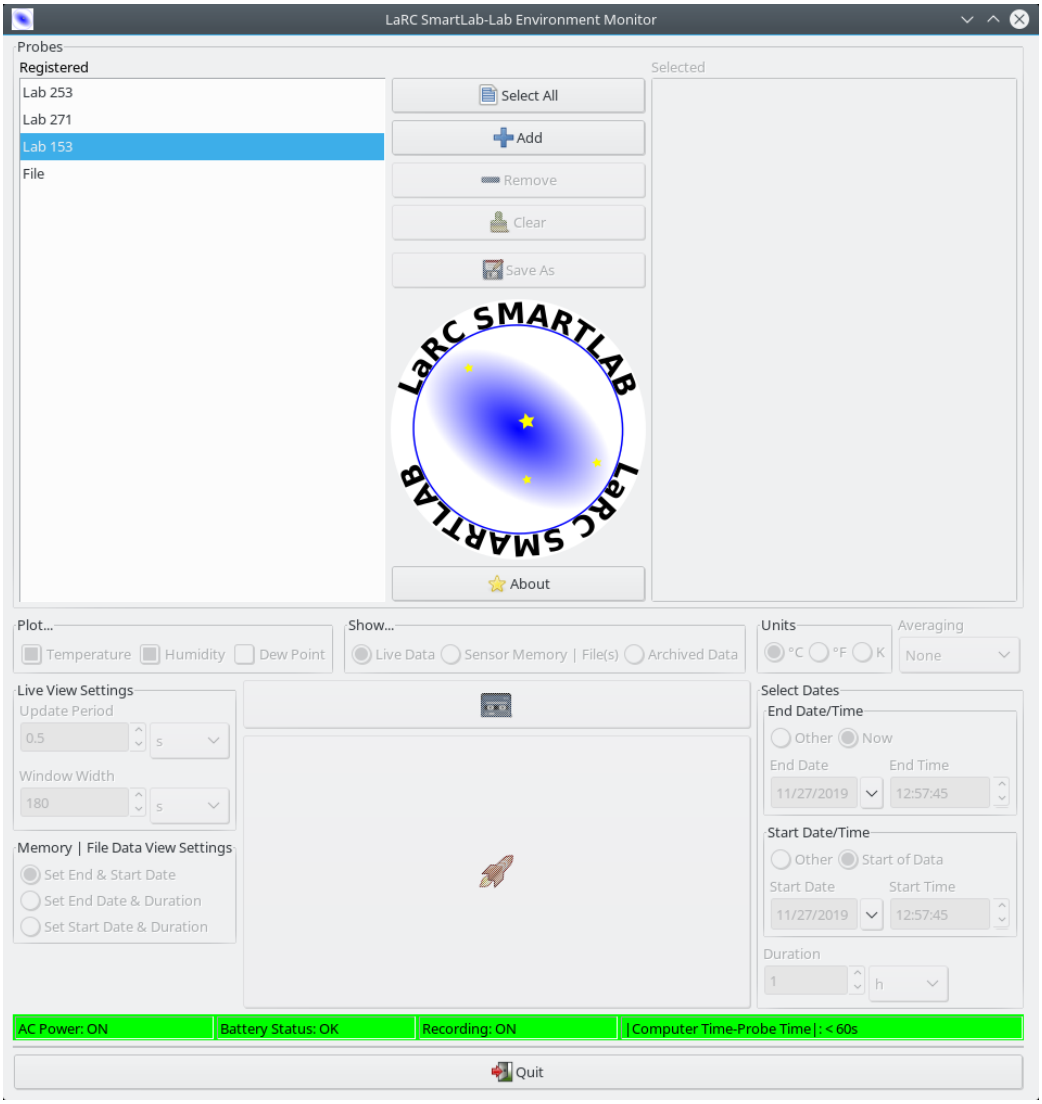
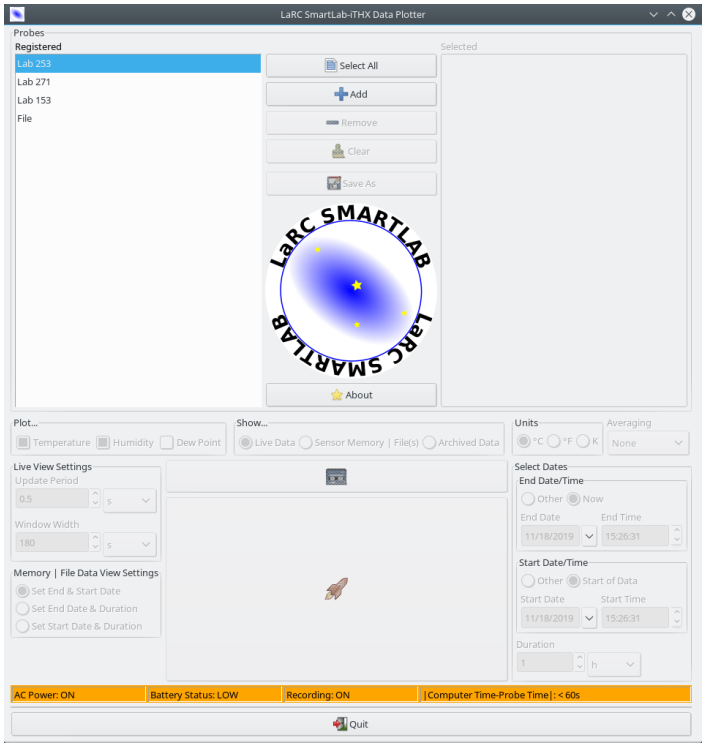
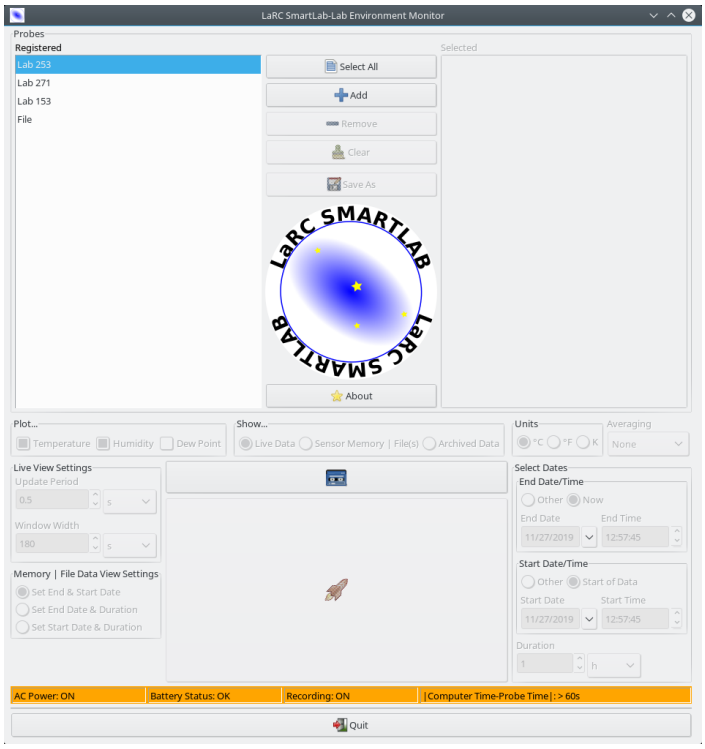


Figure 5: Probe status messages when all the parameters checked are within their accepted ranges.



(a) Low battery



(b) Probe and caller (computer) clocks out-of-sync

Figure 6: Probe status messages when one or more tracked parameters is outside of the accepted range.

3.4 Modal dialog boxes

The application interface also includes modal dialog boxes that are displayed at run-time to provide the user with additional information. An example are the dialog boxes that provide information *about* the application. The About button brings up these dialog boxes which are shown in Figure 7. The information that is displayed includes the version number of the application and the development team.

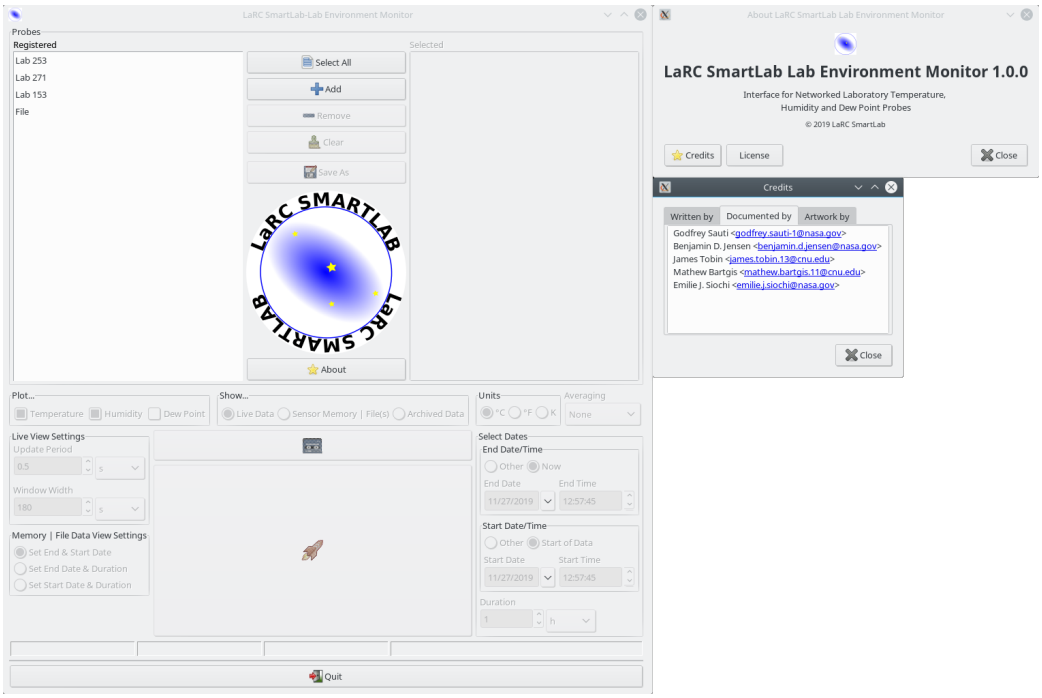


Figure 7: Dialog boxes showing information *about* the application.

4 Getting Started

When started from the terminal, in a suitably configured Python environment (see Section 6 for modules on which **lsltemp** depends), the application reads its configuration file and then checks if the probes defined in that file are online and at the addresses specified. The content and structure of the configuration file is discussed in Section 6.3. Launch of the application, in a Konsole terminal emulator running on Debian Linux, is shown in Figure 8. During the startup process, the application also gets some status information from each probe that is online. This information includes the probe's current power source, backup battery status and whether or not the probe is recording data to its on-board memory card. The information gathered for a particular probe is displayed in the status bar when the user highlights that probe in the `Registered` probes selector (Section 3.3).

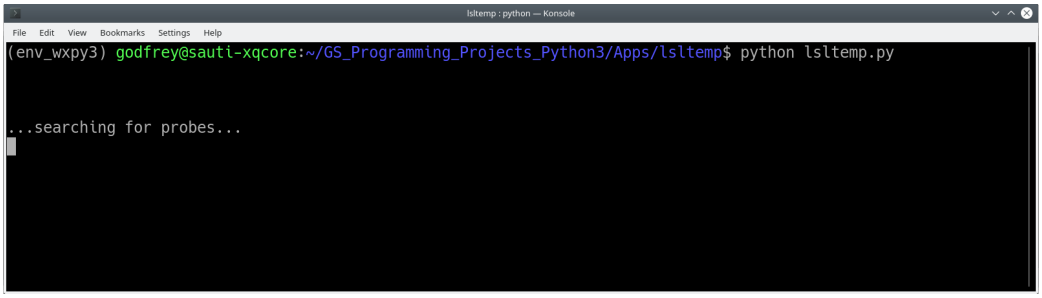


Figure 8: Application searching for probes during startup in the Konsole terminal emulator on Debian Linux.

Once the search for probes is complete the application GUI, which is shown in Figure 9, comes up. The probes defined in the configuration file may be enabled for selection or grayed out depending on whether or not they were found online. If none of the probes defined was found online, then the `Registered` probes options will include `Simulated` probes (Section 5.3). In addition to the probes found online or simulated, there is also an entry (`File`) added to the `Registered` probes column for selecting previously downloaded and saved sensor data from files for display (Section 5.4).

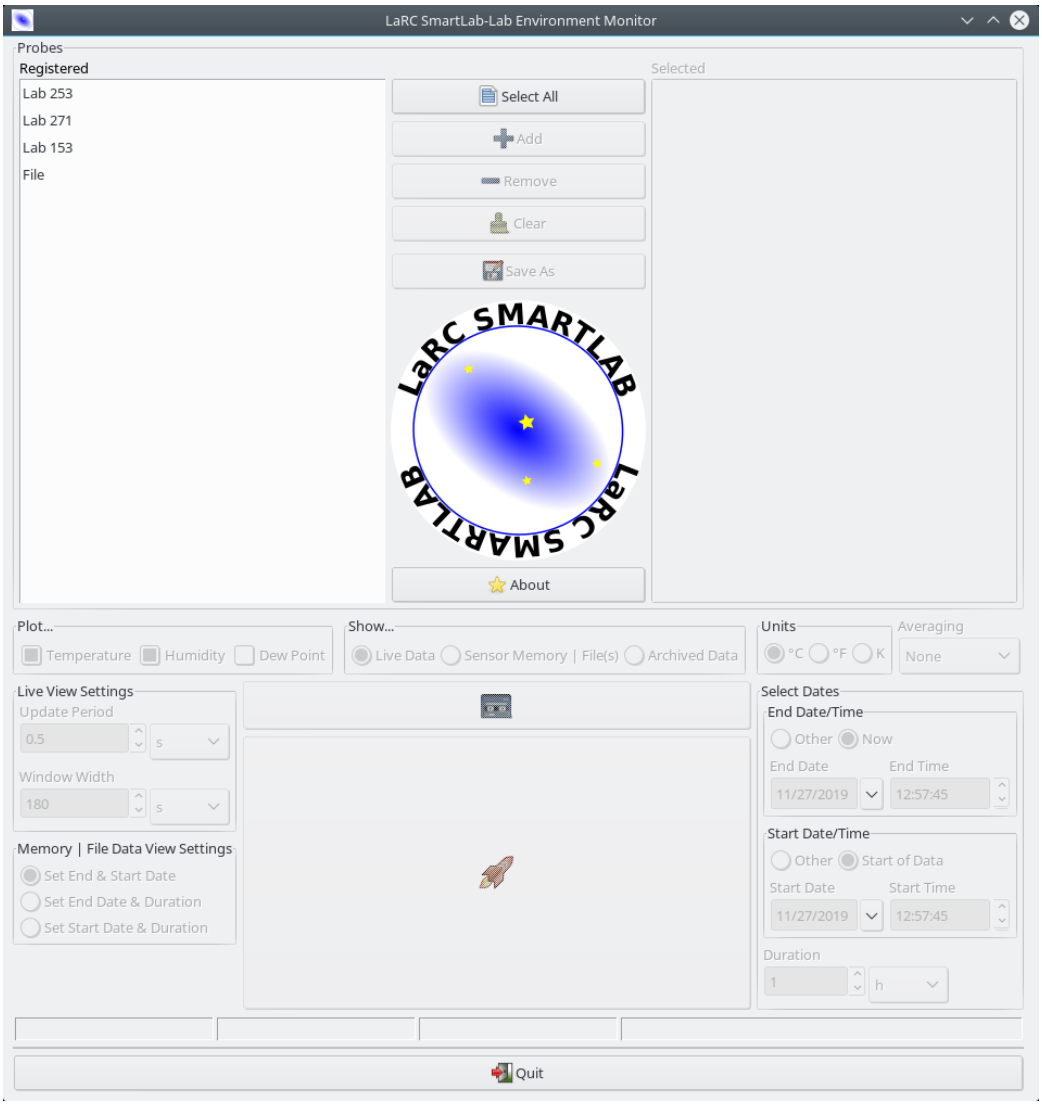
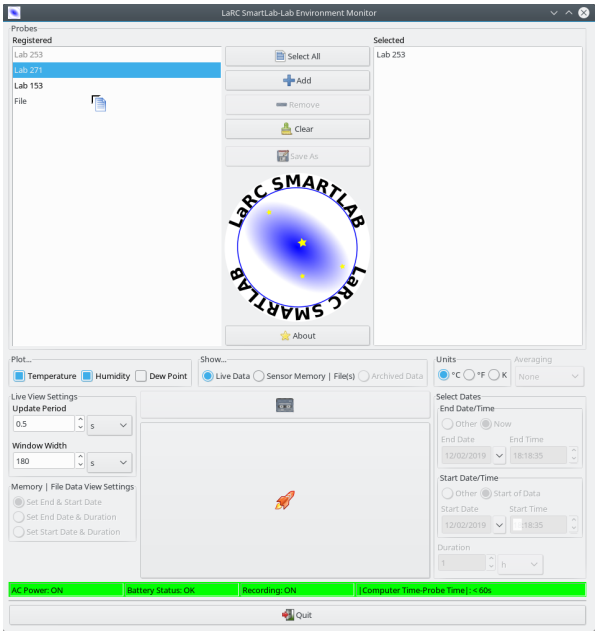


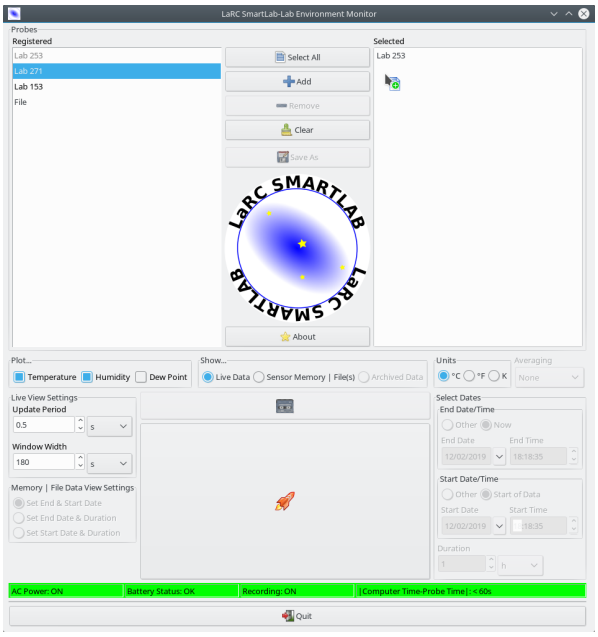
Figure 9: The **lsltemp** front panel showing a list of the probes specified in the application's configuration file and which were found to be online. Note the differences between the list of probes here to the instance when all the physical probes were offline (Figure 2).

4.1 Selecting the data to display

The application is able to display real-time (live) data or get data from a sensor's memory. Choosing the probe(s) whose data to display can be done using the selection buttons or the application's drag and drop feature shown in Figure 10.



(a) Drag start in `Registered` probes column



(b) Drop in `Selected` probes column

Figure 10: Selecting probes using the application's drag and drop feature.

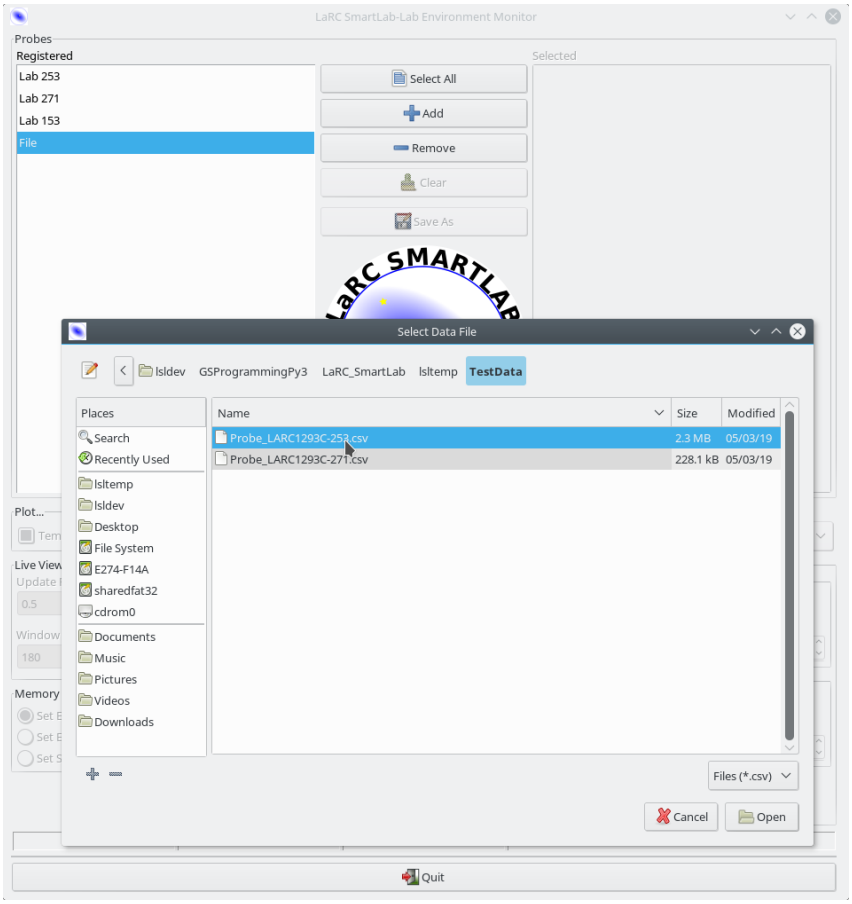
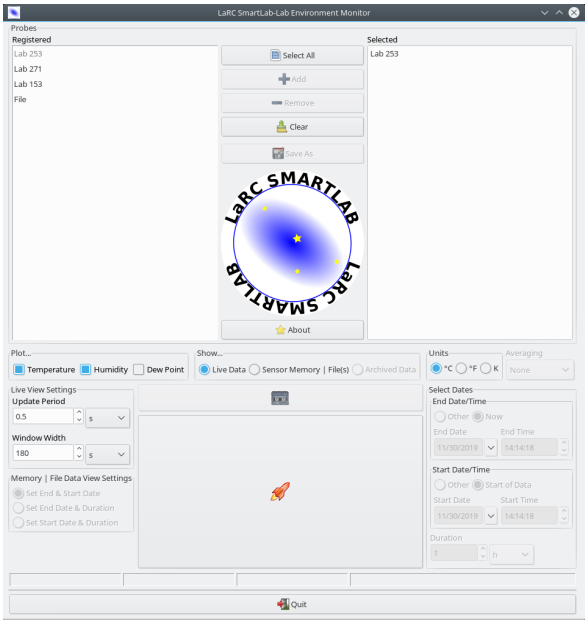
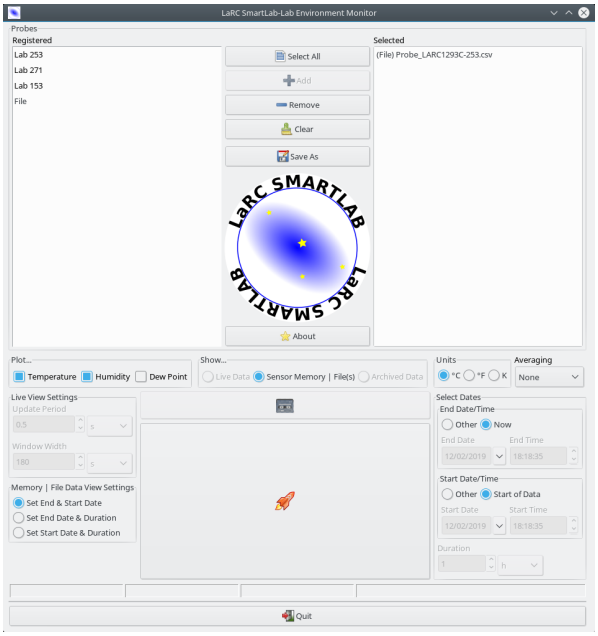


Figure 11: Dialog box for selecting data from a file for display.

Data from files can be plotted by selecting `File` in the `Registered` probes column and then using the Add button or by dragging `File` from the `Registered` probes column and dropping it in the `Selected` probes column. Adding `File` to `Selected` columns using either method will bring up the dialog shown in Figure 11. The dialog can be used to locate the file to be added. By repeating the file selection procedure, it is possible to add data from as many files as the user would like to display. Unlike for the probes, adding or dragging `File` from the `Registered` probes column into the `Selected` probes column does not cause the entry in `Registered` probes to be disabled. Note that when a file is selected, the application adds an entry to the `Selected` probes column with the name of the specific file that has been loaded, with the header `File` in front of it, as shown in Figure 12.



(a) Live streamed data plotting options



(b) Options for plotting data from sensor memory or files

Figure 12: Different controls are enabled depending on whether data will be streamed live from probes or sourced from probe memory or files.

The options that are enabled for configuring the display of data depend on whether the data will be live streamed from probes, come from probe memory or is being read in from files. The options enabled for displaying data live streamed from a probe are shown in Figure 12(a) while those for data from probe memory or files are shown in Figure 12(b).

5 Using the Application

We now present some examples of using **lsltemp** to display various data captured by the sensors or read in from files, by the application. We start by describing the display of live streamed sensor data.

5.1 Plotting live sensor data

Once one (or more) sensors have been selected, the user is able to display any combination of the Temperature, Humidity, and Dew Point. Which of the three are displayed depends the selections in the `Plot...` checkbox controls. Figures 13 and 14 show plots of different combinations of data from a single probe as well as the settings used to configure the data that is displayed.

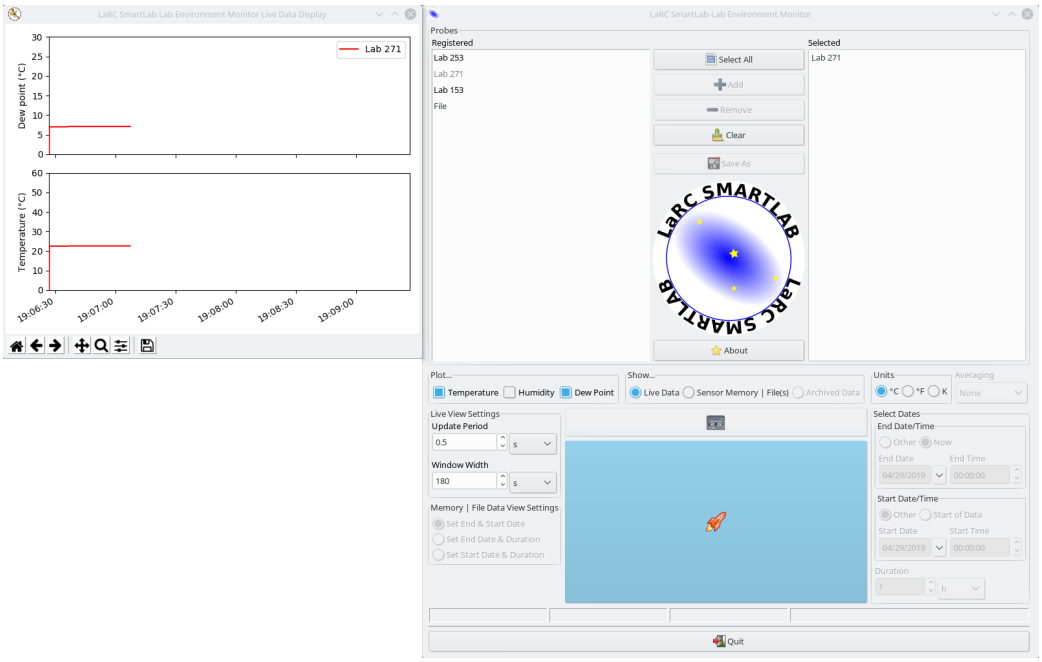
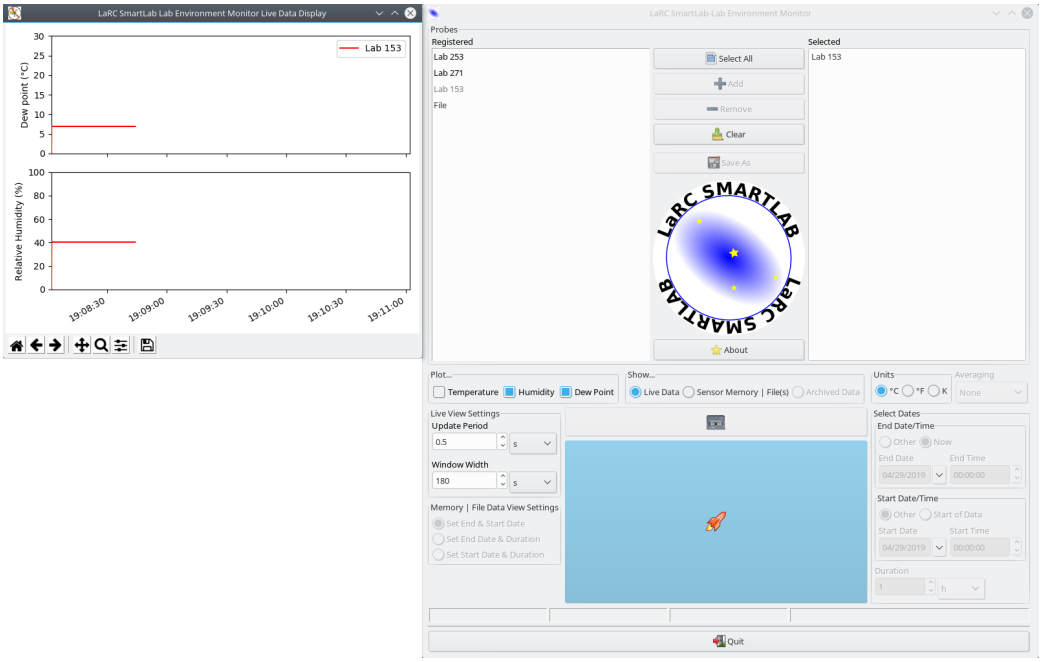
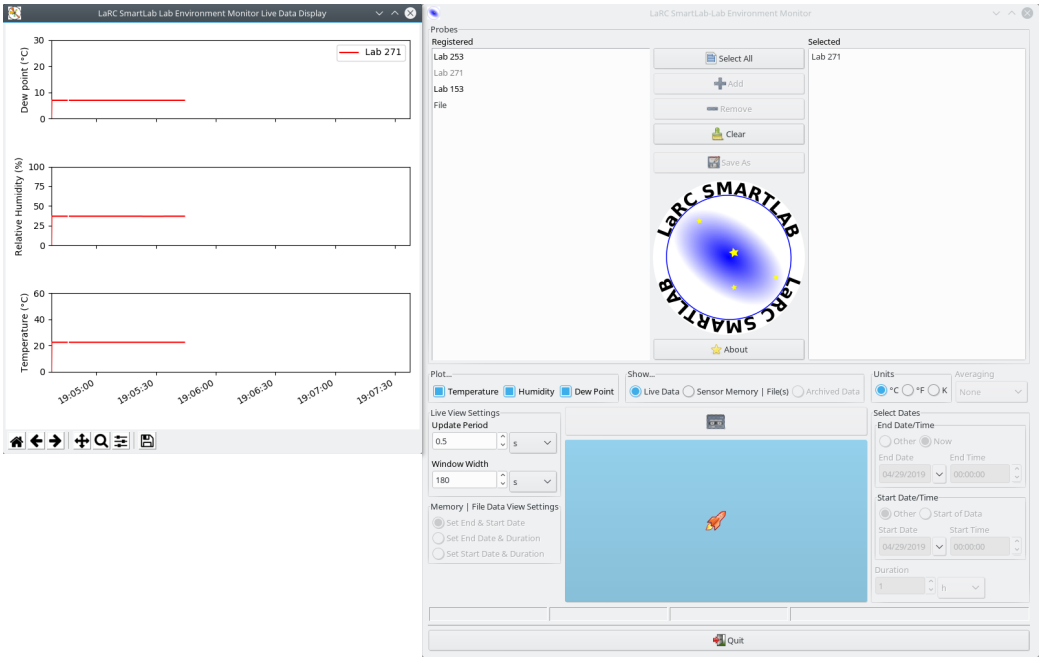


Figure 13: Plots of the Temperature and Dew Point data live streamed by a single probe.

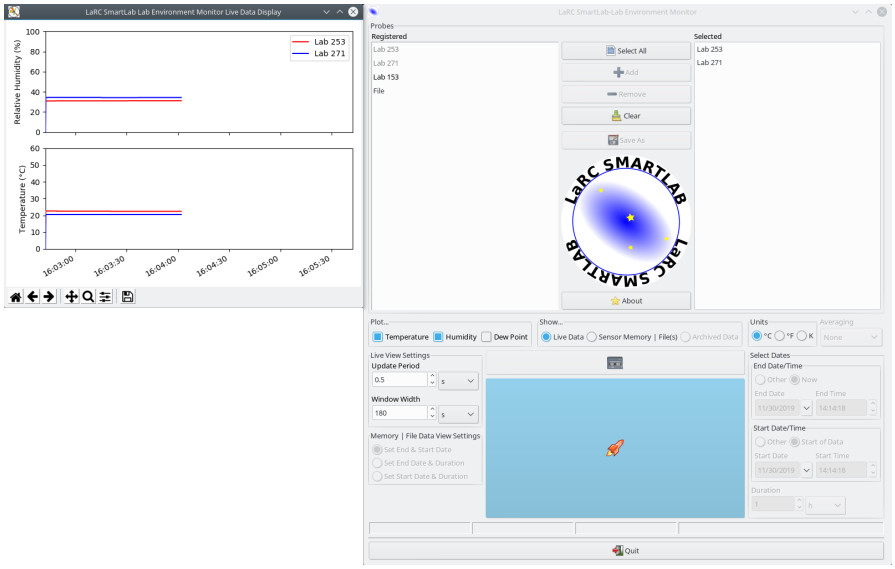


(a) Dew Point and Humidity

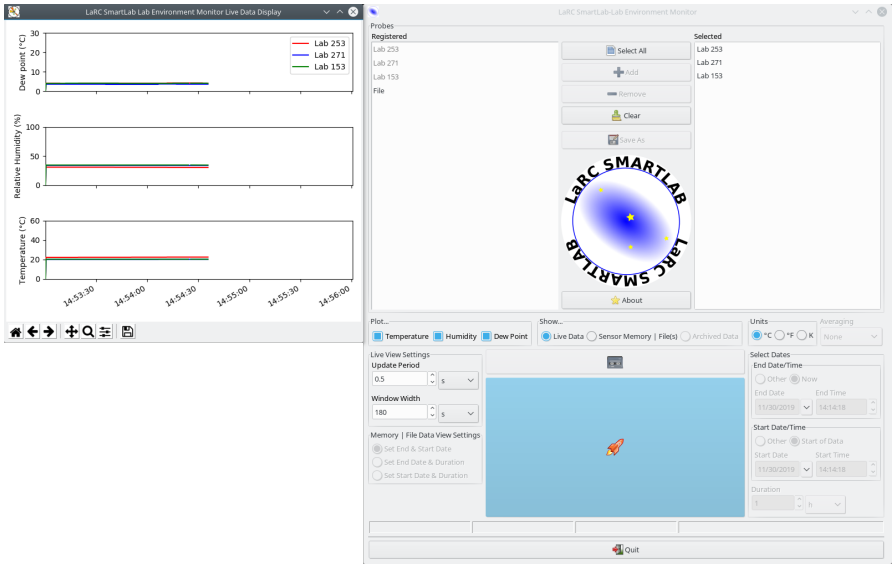


(b) Temperature, Humidity, and Dew Point

Figure 14: Displays of more combinations of sensor data live streamed by a single probe.



(a) Two probe Temperature and Humidity



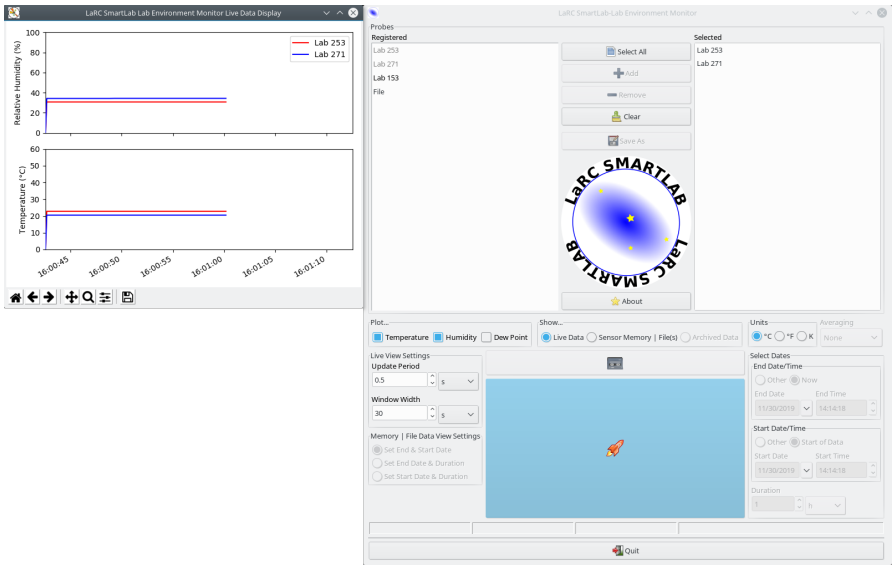
(b) Three probe Temperature, Humidity, and Dew Point

Figure 15: Plotting data live streamed by multiple probes.

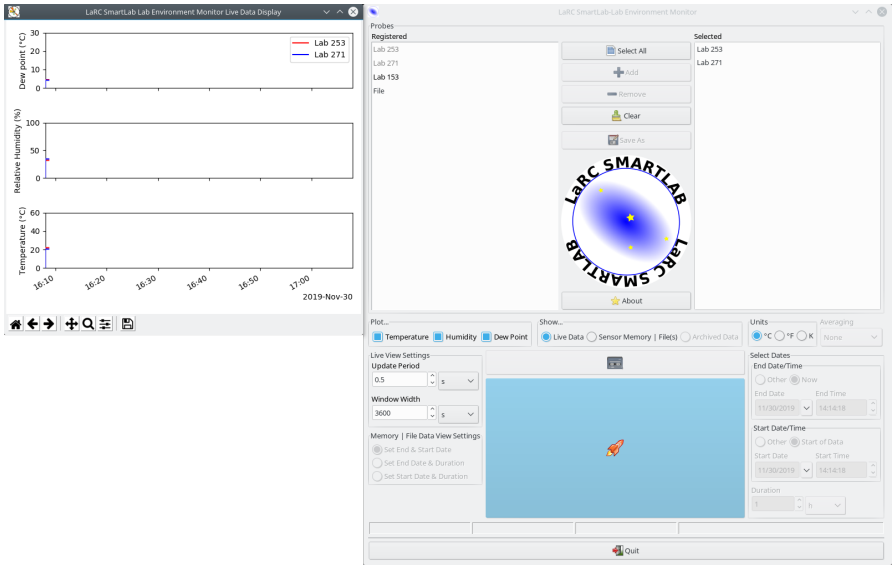
Live data from multiple probes can also be received and plotted at the same time, as is shown in Figure 15, again with different combinations of the Temperature, Humidity, and Dew Point selected using the `Plot...` checkboxes. The application is able to accommodate not receiving a data point in time from a probe by skipping that point in the plot and resuming adding the next point whenever it is received.

5.1.1 Configuring live data plots

Figures 16 and 17 show some examples of various options that are available for controlling how live data are displayed in the application. The plot window adjustments in Figure 16 control how much data are shown on the screen before an automatic rescaling of the Time axis.



(a) Shortened plot window (30 s)



(b) Lengthened plot window (3600 s)

Figure 16: Effect of changing the plot window (with update window unchanged).

The adjustment to the update period Figure 17 changes the rate at which the application requests data from the sensors. *Note that the actual rate at which data are captured will depend on the sensor's ability to respond as well as network traffic. The application is able to continue to display data with any points that have not been received from the sensor omitted from the plot.*

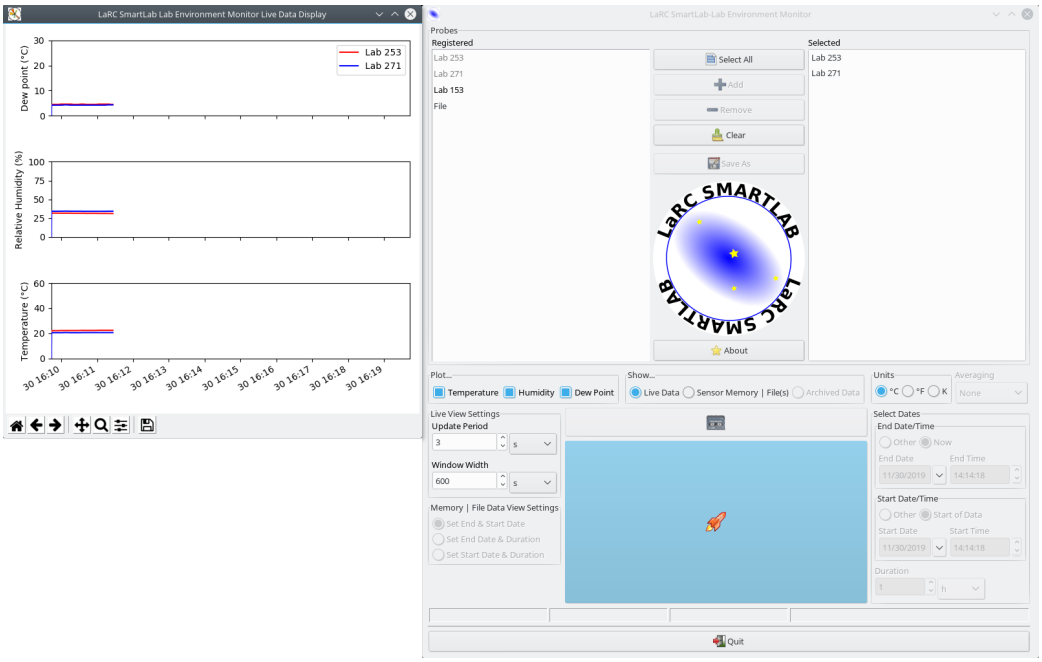
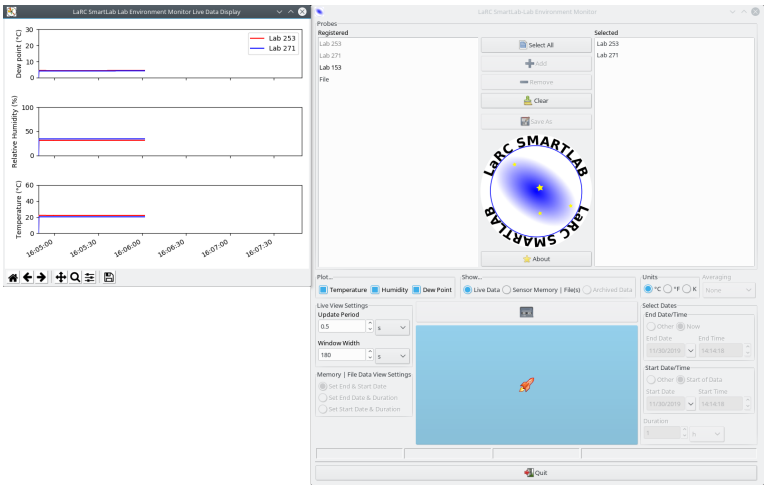


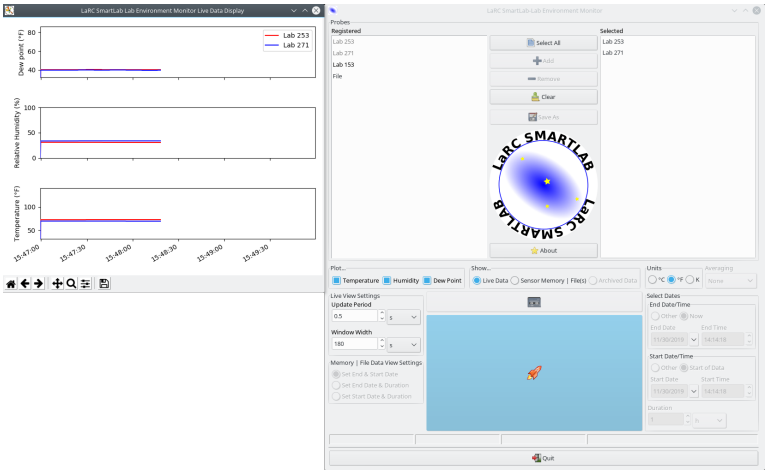
Figure 17: Changes to the update and plot windows.

5.2 Setting the display units

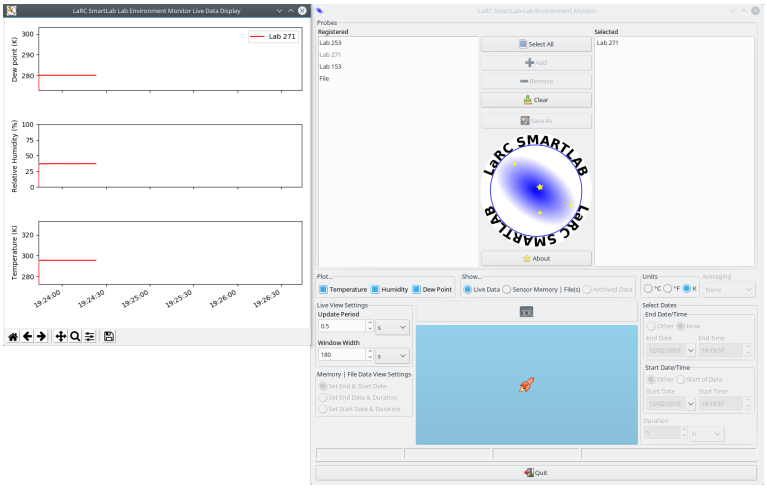
lsltemp provides options for the units in which the Temperature and Dew Point data are plotted. Any change in the units needs to be made before the data display is launched. If the user needs to change the units while data are being displayed, they can close the plot window, select the new units, and re-launch the plots. Plots with different units selected are shown in Figure 18. The selection of units is available for both live data as well as plots of data from sensor memory and previously downloaded sensor data files (Section 5.4). Live data can be and are requested from the sensors in ℃ or °F by the application depending on the user's selection of display units. For data displayed in Kelvin, the request to the sensor is made in ℃ and the conversion to K is carried out by the application. Any downloads of data from the sensor to files are in ℃ and all units conversions for data from files or sensor memory are carried out by the application (See Section 6.2.1 for more on the implementation of units conversion in the application).



(a) ℃



(b) °F



(c) K

Figure 18: The data can be plotted using different units for the Temperature and Dew Point.

5.3 Offline plotting with `Simulated` probes

When **lsltemp** finds that **all** the physical probes are offline during startup (Figure 19), it will create a set of `Simulated` probes as shown in Figure 20. These allow interaction with the application's user interface and access to most, but not all, the functions available when physical probes are online. This feature, which is in addition to being able to plot data from files (Section 5.4), gives the user additional opportunities to familiarize with the application interface. The use of `Simulated` probes in this and other applications also gives developers the ability to test and debug most of the functionality of the applications, even when they do not have access to the hardware in the laboratory.

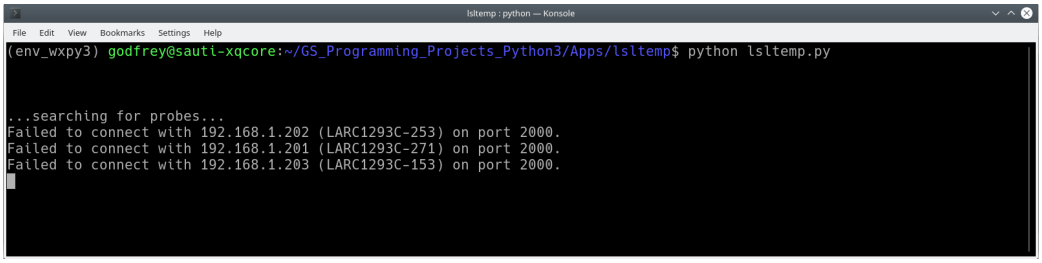


Figure 19: The startup of the application in the terminal and the search for probes probes listed in the configuration file. In this case, none of the those probes are online.

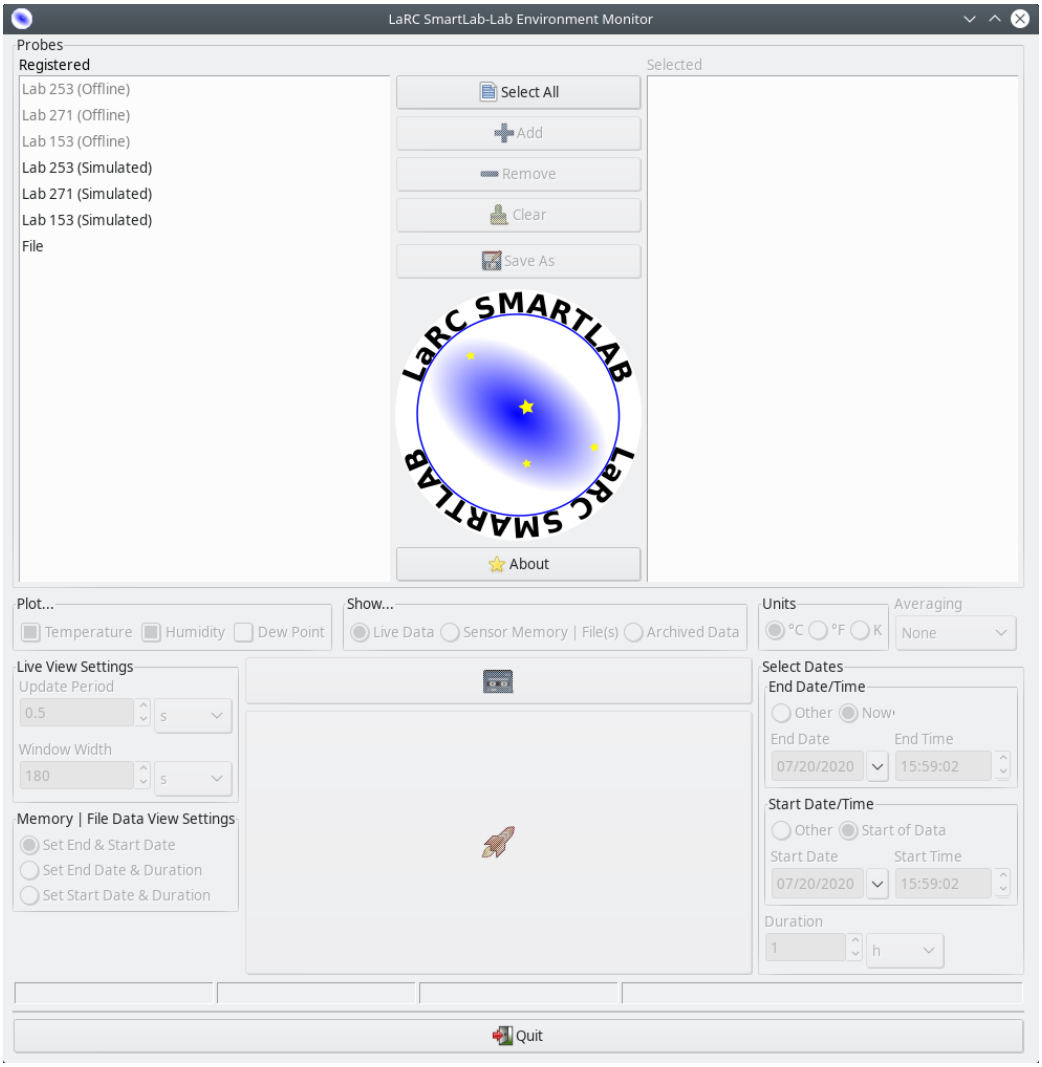
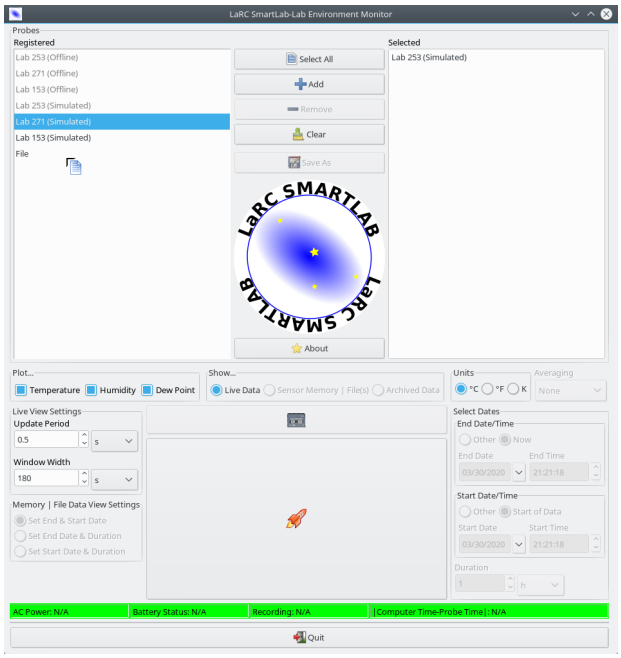
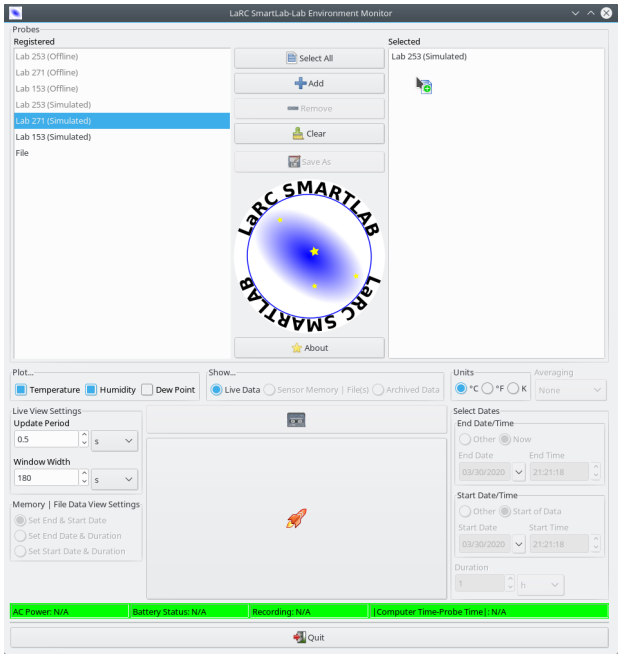


Figure 20: **lsltemp** user interface with the `Simulated` probes created when all the physical probes are offline.

The user is able to select data to plot from the `Simulated` probes in a way similar to the physical probes, for example using the drag and drop feature as shown in Figure 21.



(a) Drag start



(b) Drop

Figure 21: Selection of `Simulated` probes is similar to that of physical probes. Here, the drag and drop functionality for `Simulated` probe selection is shown.

The plot options that are available for `Simulated` probes are shown in Figure 22. Note that there are some functions that are only compatible with physical probes which are grayed out. These include getting data from sensor memory. Unlike the physical probes, the `Simulated` probes *have no memory*. Therefore, displaying data from both `Simulated` probes and files at the same time is not supported (Section. 5.4.3).

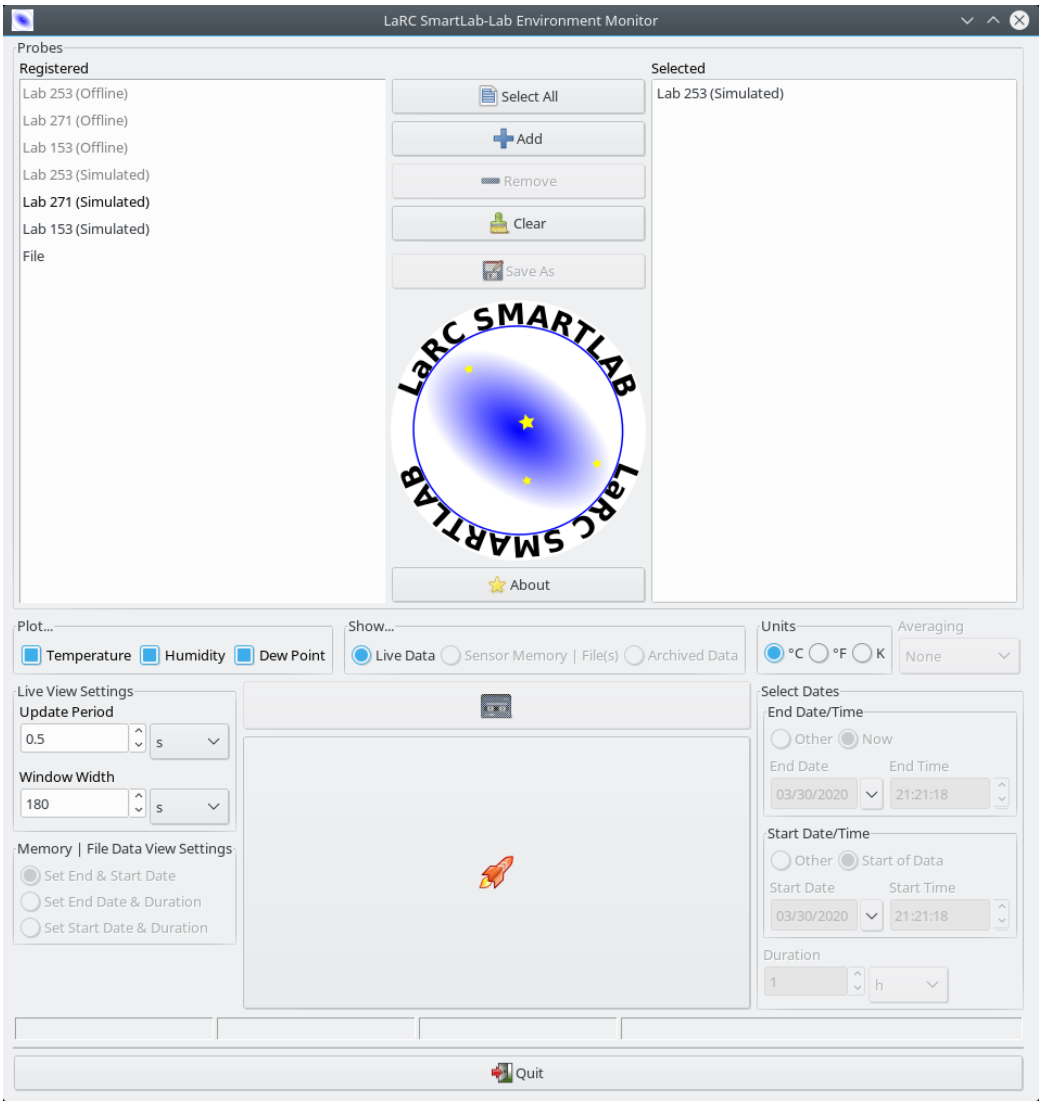


Figure 22: The plot options available for the data from `Simulated` probes can be seen by the controls that are not grayed out.

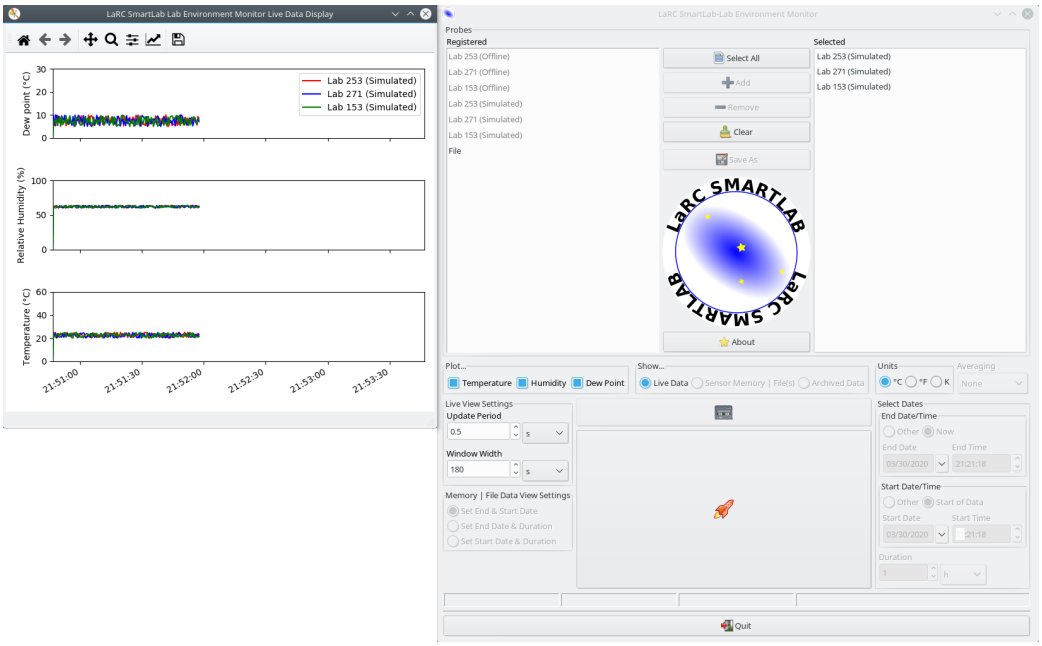
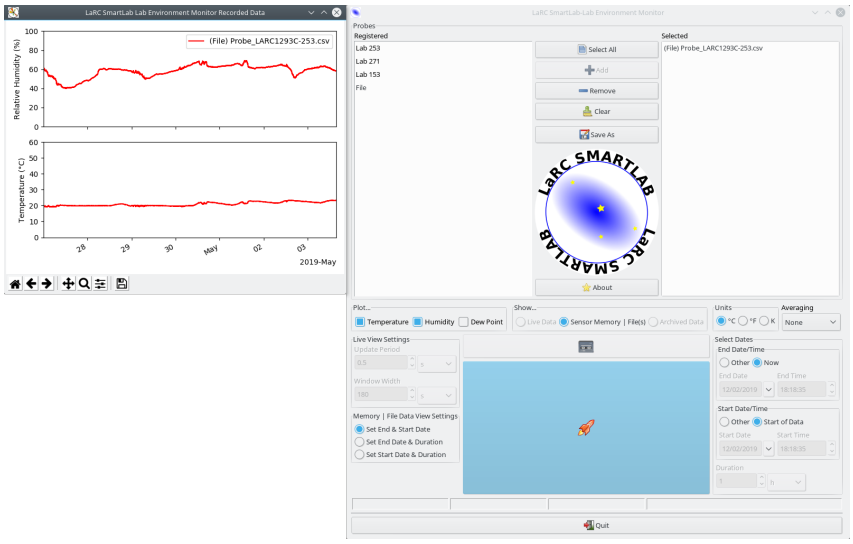


Figure 23: Plots of data "live streamed" from `Simulated` probes.

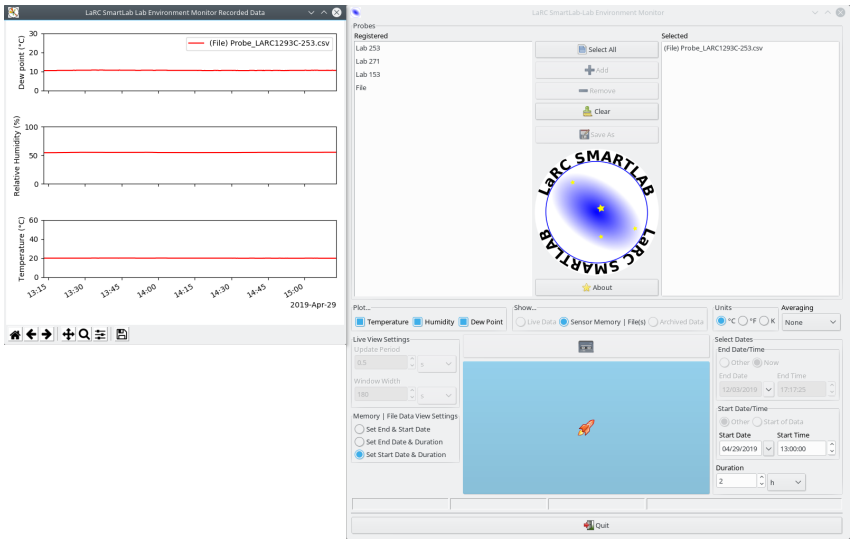
Figure 23 shows plots of data "live streamed" from multiple `Simulated` probes. Using these, the user can explore the effects of changing the plot windows, the units, the data that is selected for display and other features of the application.

5.4 Plotting data from sensor memory or files

In addition to data live streamed from probes that are online (or `Simulated` probes, when all the physical probes are offline), the application can plot data from sensor memory or from files containing previously downloaded sensor data. For the user's convenience, **lsltemp** allows the range of dates for the data that is displayed to be specified in different ways as shown in Figure 24. The options for setting the date range include specifying the start and end date (Figure 24(a)) or the start date and duration (Figure 24(b)).



(a) Start and end date



(b) Start date and duration

Figure 24: The application provides the user with multiple options for selecting the range of dates for data that will be displayed from sensor memory or files.

For added flexibility in selecting the plotting range, **lsltemp** includes options for the start and end dates to be automatically determined from the data that in the files or probe memory as seen in Figure 25. The end date/time option "Now" is used when the user requires the application to fetch all data saved in memory from some start data to the present moment and also when requiring the same from data that is in a file. For data that is being plotted from a file, requesting the date range to be from some start date up until "Now" will return data up to the end of that file.

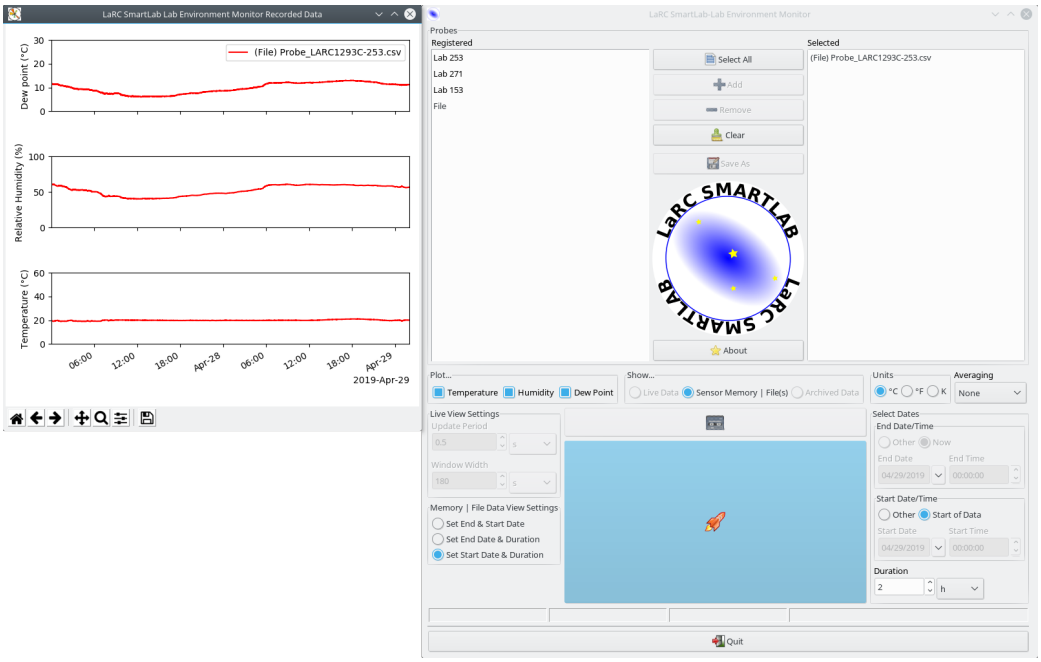


Figure 25: The application includes options for the date ranges to be determined at display time from the data that is available in sensor memory or in the file(s). For example here, the range selector is set to be a start date and a duration, with the start date determined from the data in the files read into **lsltemp**.

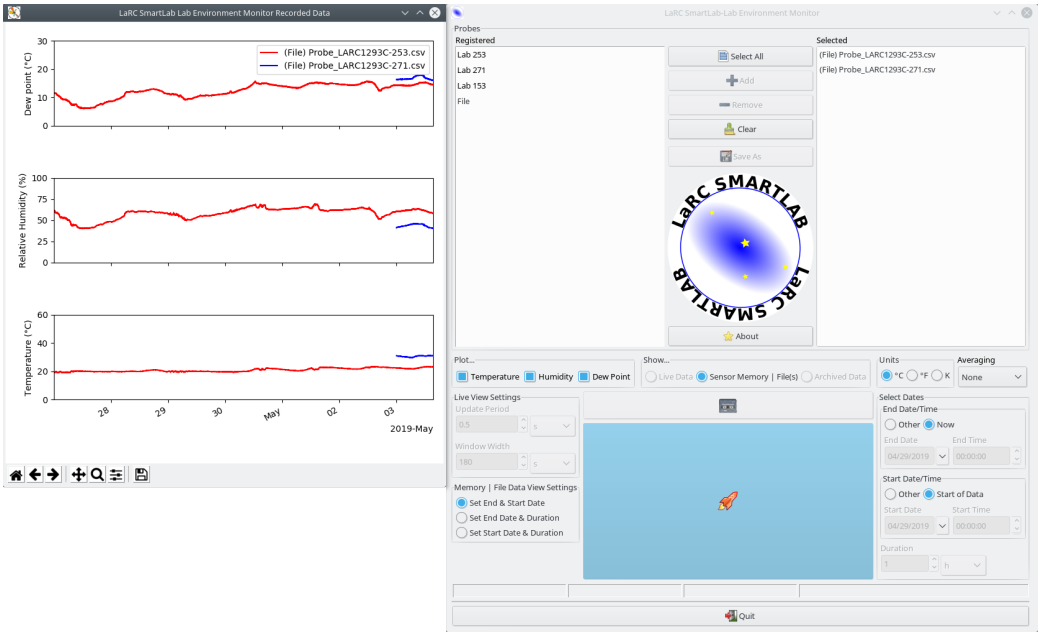
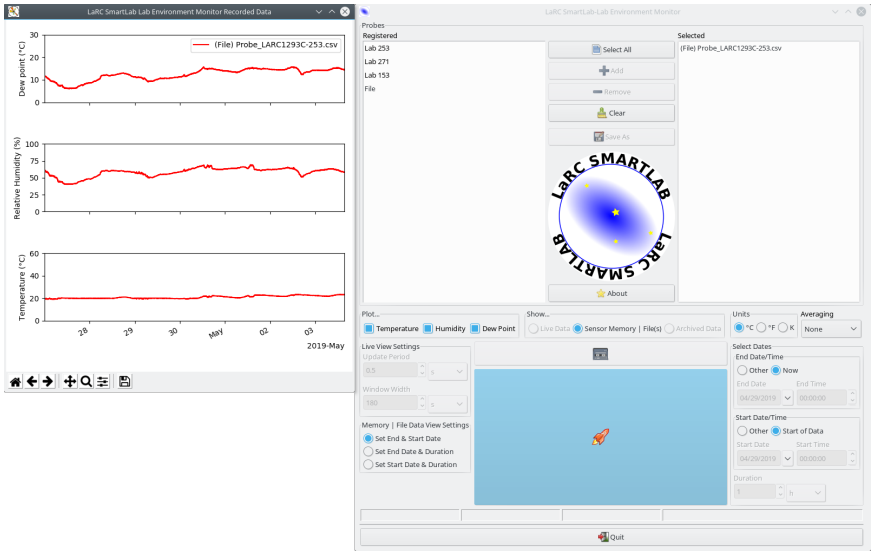


Figure 26: When sourcing data from multiple files, the application will plot the data from each file that is in the selected date/time range.

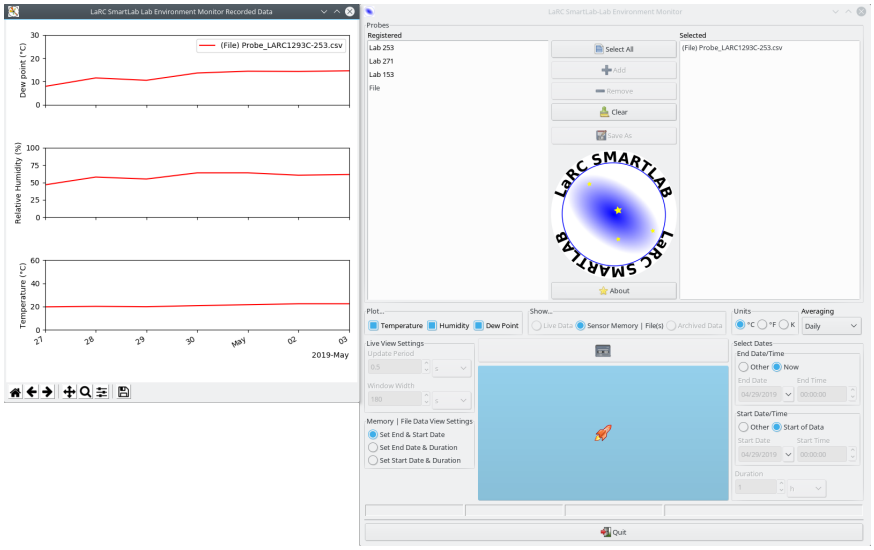
The user can select to plot data from multiple files as shown in Figure 26. Any data in each file that matches the selected date range will be plotted.

5.4.1 Averaging

For large datasets, it may be advantageous to apply some time averaging to the readings before displaying the data to the user. This allows for trends in the data to be highlighted on per minute, hourly, daily, and monthly time scales. Figure 27 shows data displayed without (Figure 27(a)) and with averaging (Figure 27(b)). The averaging, for example daily, is accomplished by an **lsltemp** helper function that intercepts the data from sensor memory or file for pre-processing before sending it to the data display function.



(a) No averaging

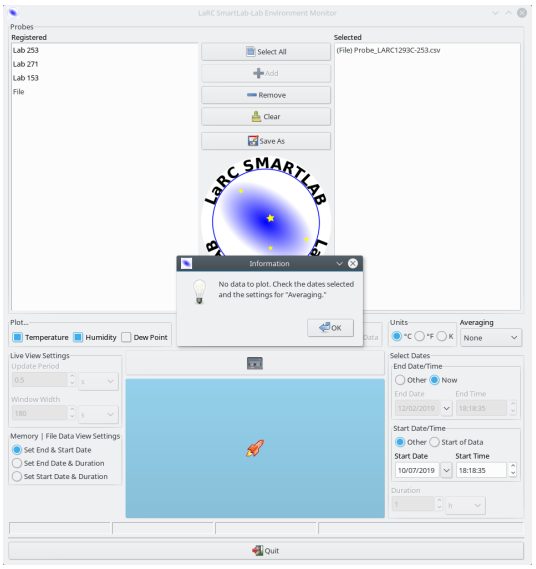


(b) With daily averaging

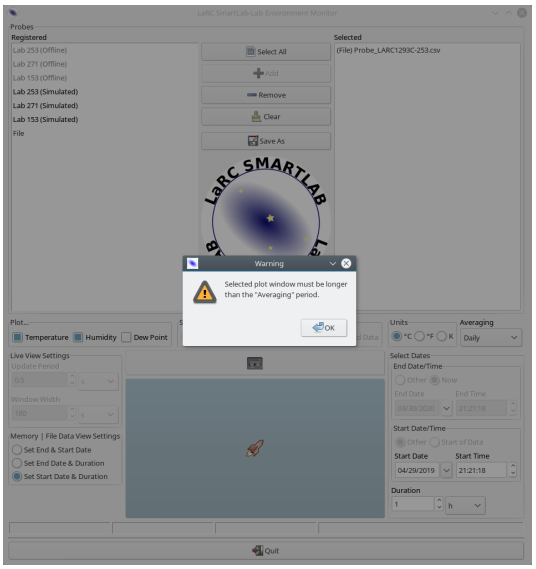
Figure 27: Data plotted without and with averaging. The use of averaging can help to make trends in the data clearer to see at the relevant time scale.

5.4.2 No data to plot

There are situations when there will be no data left to plot after the user's display date range and averaging options have been applied to data loaded from file or sensor memory as shown in Figure 28. A simple case where there will be no data to plot is when the date range selected lies outside of the range of the data that is in the file(s) or sensor memory (Figure 28(a)). User settings for averaging may also conflict with the selected date range leaving no data available for plotting (Figure 28(b)).



(a) No data in selected range

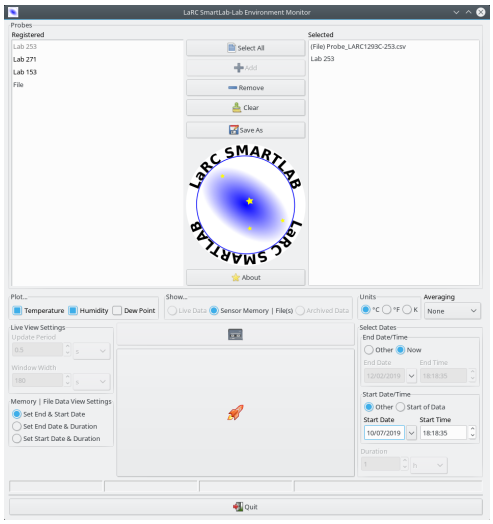


(b) Conflict between data range and averaging

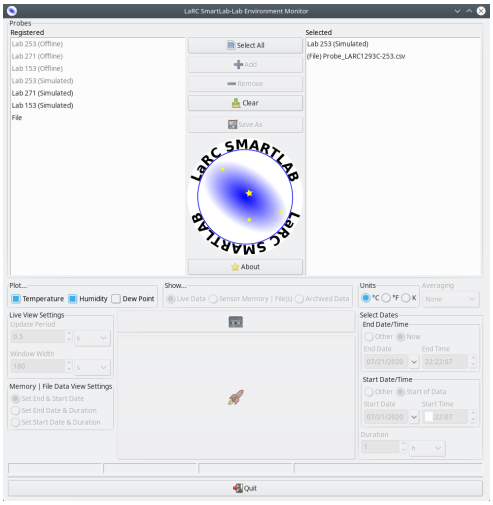
Figure 28: The selected date range or averaging settings may leave no data available to display. The application will alert the user with an appropriate message.

5.4.3 Data incompatibility

Some sources of data cannot be consistently plotted together by the application. **lsltemp** will prevent the user from trying to make incompatible plots (Figure 29). When a file and a physical probe that is online are selected, the only data from the probe that can be plotted is from it's on-board memory. Therefore, the *Show...Live Data* selector is disabled and the control defaults to the option for getting data from probe memory (Figure 29(a)). Plotting is disabled altogether when a file and a `Simulated` probe are selected (Figure 29(b)).



(a) Live physical probes and file data

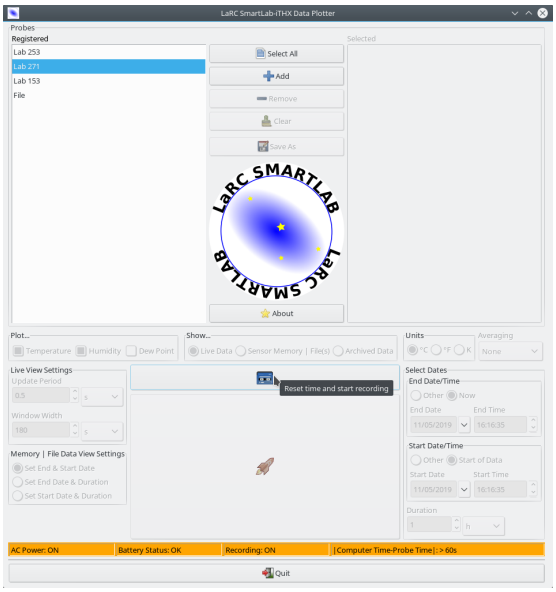


(b) `Simulated` probes and file data

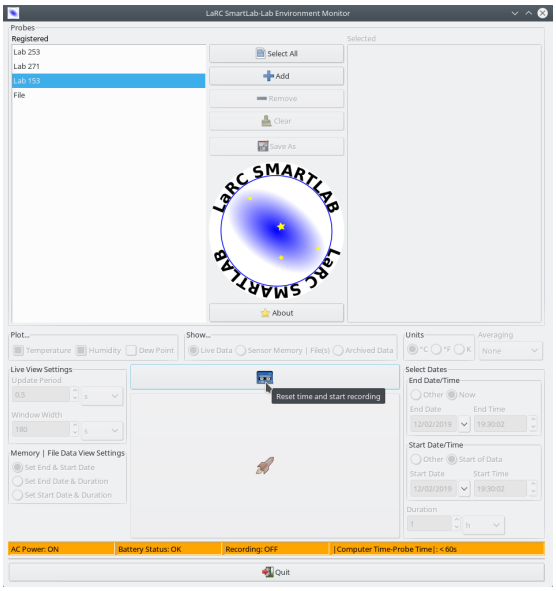
Figure 29: The application disables some options to prevent attempts to make plots using incompatible settings. Note: Unlike the physical probes, `Simulated` probes "have no memory" and therefore there is no data from them that can be plotted at the same time as data from files.

5.5 Resetting the probe clock and starting recording

We now discuss how to reset the sensor clock (to the time provided by the calling computer) and to (re)start data recording to the sensor's on-board memory. The Start Recording button is enabled if a highlighted probe's clock is out-of-sync with the caller or recording to on-board memory is off, as shown in Figure 30.



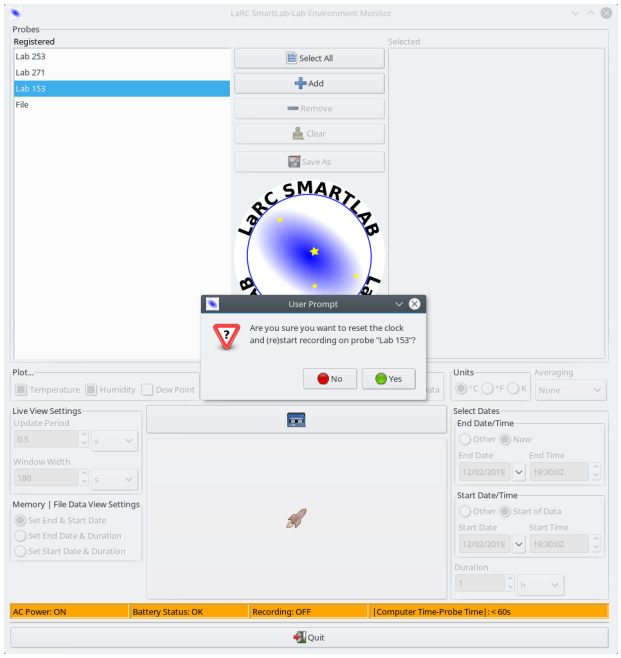
(a) Sensor and caller clocks out-of-sync



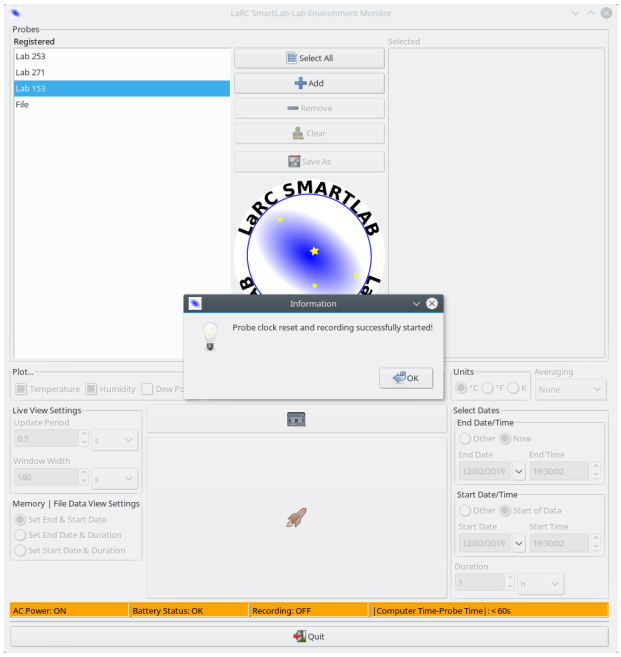
(b) Recording off

Figure 30: The clock reset and (re)start recording button is enabled when the probe's internal clock is out-of-sync with the caller's clock or data are not being recorded to the probe's on-board memory.

Pressing the Start Recording button brings up dialog boxes that will (1) stop recording if it is running, (2) reset the probe time to the caller's time, (3) start recording again. These dialog boxes are shown in Figure 31.



(a) User prompt



(b) Reset recording start successful

Figure 31: Resetting the probe clock and (re)starting recording of data to the sensor's on-board memory.

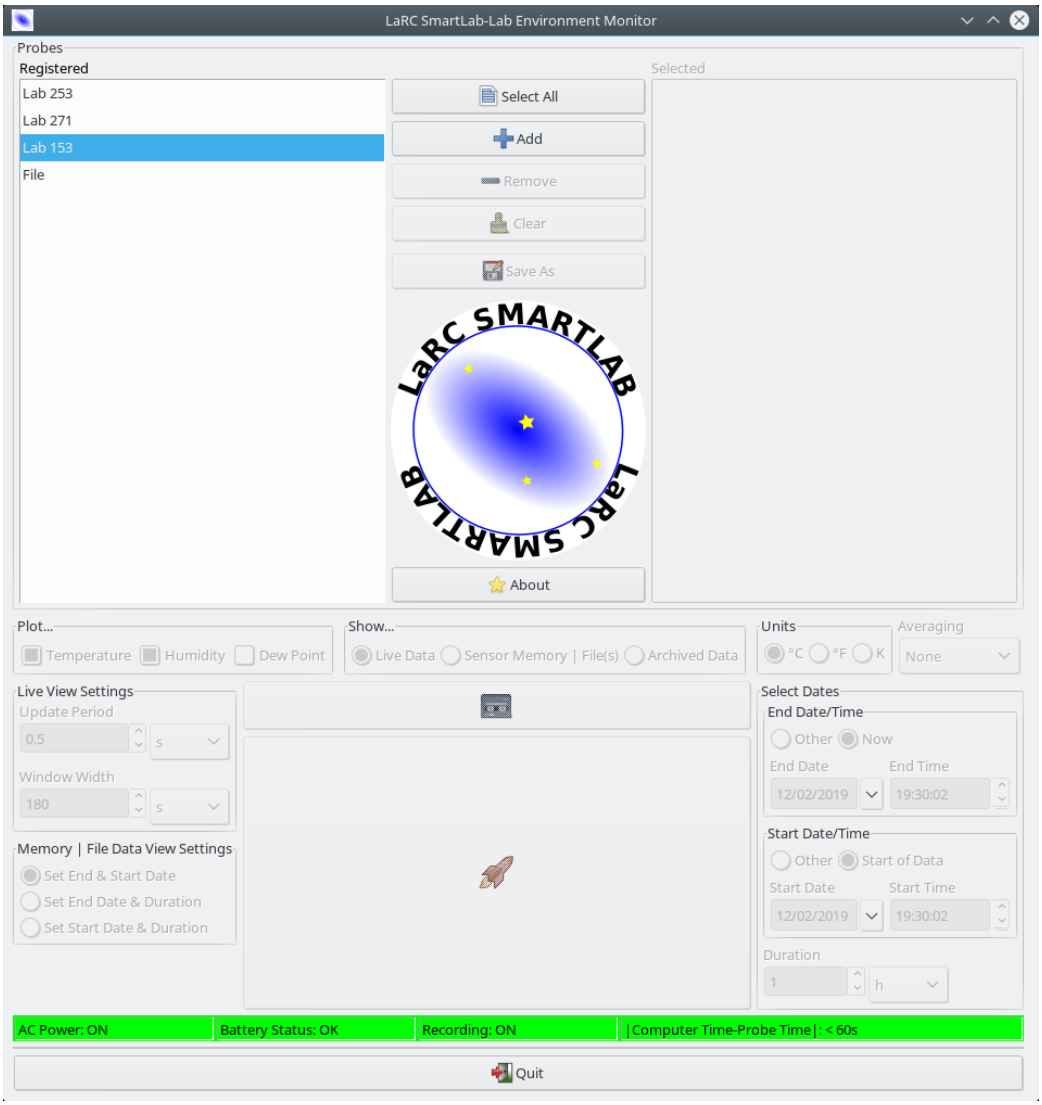


Figure 32: User interface after a reset of the clock and (re)start of recording.

Upon reset of the clock and restart of recording, the application requests status information from the probe. Figure 32 shows the user interface following a successful reset of the clock and restart of recording. The green status bar shows that AC power is on, the battery charged, the time on the probe's internal clock is in-sync with that on the caller's clock (to within 60 s) and that the probe is recording data to it's on-board memory.

6 Implementation

The application is implemented as a series of modules in order to aid code maintenance and reuse. The key elements of these modules are briefly described in this section. Note that the description provided is not meant to be a full listing of the code. It is instead meant to provide pointers to developers that wish to work on similar projects or to use some of the tools and libraries utilized here.

6.1 Top level

Listing 1: Import of required modules.

```
13 from pathlib import Path, PurePath
14 import json
15 import datetime
16 import wx
17 from pandas.plotting import register_matplotlib_converters
18 import libgsalt.barebonesunits.barebonesunits as bbunits
19 import libgsalt.gui.labelledcontrols as lc
20 import libgsalt.gui.miscdialogs as miscdialogs
21 import libgsalt.gui.icons as icons
22 import lsltempbase
23 import lsltempdatetime
24 import lsltempcontrols
25 import lsltempdatatools
26 import lsltempplots
27
28 register_matplotlib_converters()  # required by the pandas dataframe
```

The top level **lsltemp** application calls in a number of standard Python modules such as **pathlib** and **datetime**, custom modules that provide a range of common tools for multiple projects (**libgsalt.***) and modules which provide functionality specific to this application (**lsltemp...**) (Listing 1).

The standard Python modules imported are:

1. **pathlib:** Provides tools for managing files[1]
2. **json:** Javascript Object Notation handler[1]
3. **datetime:** Classes for manipulating dates and times[1]
4. **wx:** GUI toolkit for python[2]
5. **pandas:** Data analysis and manipulation tool[4,5]

The **libgsalt.*** library modules which provide GUI and data processing functions are:

1. **libgsalt.barebonesunits:** Defines units for various quantities and provides fast conversion between different units

2. **libgsalt.gui.labelledcontrols:** Combines wxPython widgets such as a *numeric control* (wx.SpinCtrl), a *label* and a *caption* (wx.StaticText) into a single code element. **labelledcontrols** allows the control together with its label and caption to be handled as one entity when placing them into the calling application code (the **lsltemp** application in this case). This helps to simplify and shorten the code of the application using the **labelledcontrols** object. The elements are laid out in a *sizer* (such as a wx.FlexGridSizer) by the module with only a simple instruction required, in the calling application, for the positioning of the label and caption relative to the control. Figure 33 shows examples of the layout of the control, label and caption provided by the module. **labelledcontrols** also includes builtin tools for the validation and acceptance of the data input into the controls.

3. **libgsalt.gui.miscdialogs:** Provides standard dialog boxes such as the about dialog box (Section 3.4, Figure 7)

4. **libgsalt.gui.icons:** Provides the icons used in the application packaged as Python code to make them more portable

The top module consists of the two classes shown in Listing 2 and the initialization function in Listing 3.

Class **ProbeDropTarget** provides the application drag and drop capabilities used when selecting probes.

The application widgets are defined in the class **MainWindow** and its parent class **lsltempcontrols.MainWindowControls** (Section 6.2.3).

Listing 2: Classes.

```
42 class ProbeDropTarget(wx.TextDropTarget):
43     """
44     The class provides an interface for text drag and drop.
45 
46     Extends wx.TextDropTarget.
47 
48     """
49 
50     def __init__(self, target_control_for_drop):
```

```
89 class MainWindow(lsltempcontrols.MainWindowControls):
90     """
91     The class provides the main interface for the application.
92 
93     Extends lsltempcontrols.MainWindowControls
94     """
95 
96     def __init__(self, parent, _id, title):
```

The **lsltemp** initialization function described in Listing 3 is called during startup of the application (Listing 4). The application launches as a **wx.App**[2].

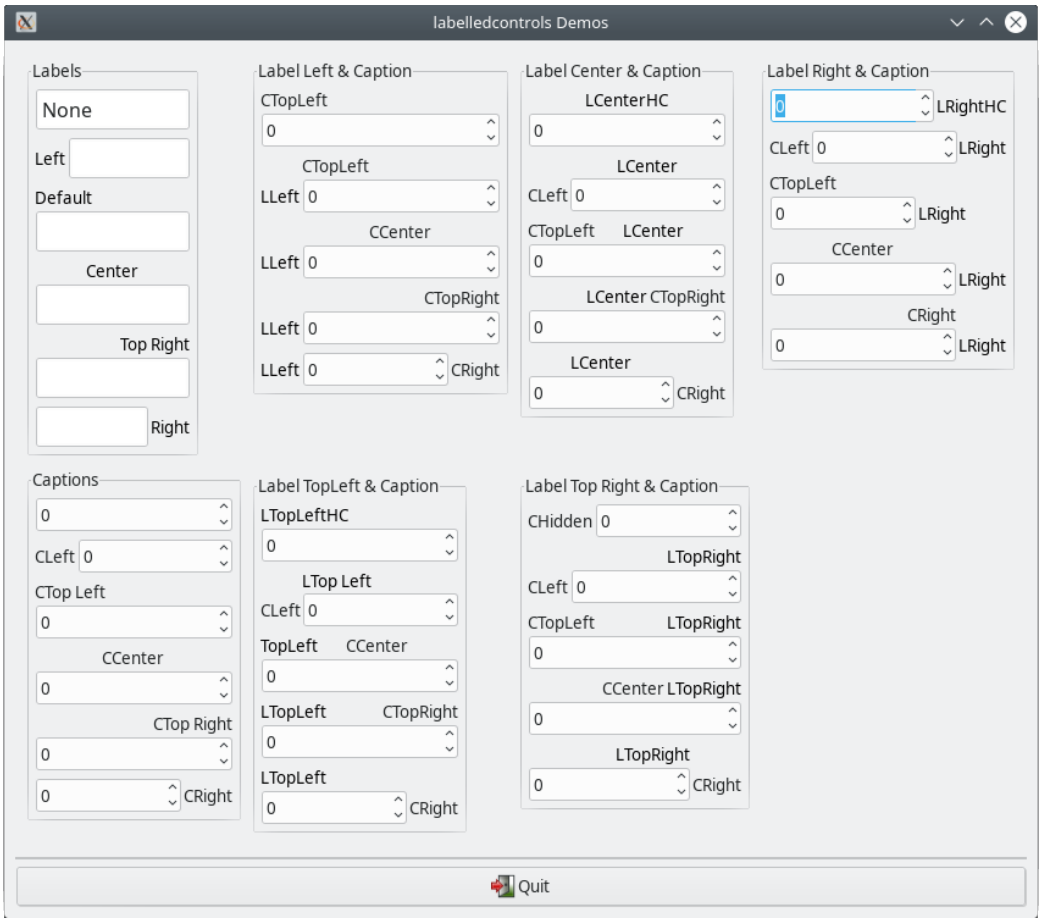


Figure 33: The controls for the **lsltemp** user interface are provided by the module **libgsalt.gui.labelledcontrols**. This module enables the control together with its label (L) and caption (C) to be handled as a single component when placed into the main application code. The placement of the labels and captions, relative to the controls, can be specified simply as “Left,” “TopLeft,” “Center” and so on.

Listing 3: Initialization function.

```
802 def initialize_the_application_main_window():
803     """
804     Create the application's main window.
805 
806     Returns
807     -------
808     None.
809 
810     """
811     frame = MainWindow(
812         None, wx.ID_ANY, 'LaRC SmartLab - Lab Environment Monitor')
```

Listing 4: Application startup.

```
817 app = wx.App(False)
818 initialize_the_application_main_window()
819 app.MainLoop()
```

6.2 Application specific supporting modules

Additional modules that are specifically tailored for this application **lsltemp**.... These are briefly described below.

6.2.1 lsltempbase

Modules for reading measurement data from instruments and files (**libgsalt.instrument.***) are imported into the **lsltemp** app by the base module (Listing 5). A number of constants used by the application are also defined in the base module (Listing 6).

Listing 5: **lsltempbase** imports.

```
26 import libgsalt.instrument.ithxtemperaturehumidityreaderbase as ithxbase
27 import libgsalt.instrument.ithxtemperaturehumidityfilereader as ithxfilereader
```

Listing 6: **lsltempbase** constants.

```
29 SHOW_LIVE_DATA = -1
30 SHOW_MEMORY_OR_FILE_DATA = 0
31 SHOW_ARCHIVED_DATA = 1
32 
33 NO_AVERAGING = ithxbase.NO_AVERAGING
```

To maintain a consistent interface for units conversions throughout the **lsltemp** application, **lsltempbase** references an instance of the **libgsalt.barebonesunits** converter which is instantiated in the base module for interfacing with iTHX probes, **libgsalt.instrument.ithxtemperaturehumidityreaderbase** (Listing 7). Each of the other modules in the application in turn point to the units converter reference in **lsltempbase**.

Listing 7: Initialization of the units converter.

```
35 units_converter = ithxbase.units_converter
```

Only one "public" function, for obtaining data averaging frequencies, is defined in the base module (Listing 8).

Note that the Python language does not have strictly private functions.

Listing 8: **lsltempbase** functions.

```
38 def get_data_averaging_frequencies():
39     """
40     Obtain time window options to be used for averaging data.
41
42     Returns
43     -------
44     `dict`: libgsalt.instrument.ithxfilereader averaging time consts
45     Options for the frequencies to use in memory and file data averaging.
46
47     """
48     return ithxfilereader.get_averaging_frequencies()
```

6.2.2 lsltempdatetime

This module provides constants and functions used for managing the date and time settings for data capture and plotting. Two standard Python modules[1], for handling the date and time, are imported into **lsltempdatetime** (Listing 9). A number of constants are defined in the module and shown in Listing 10. Only one "public" function is defined in the module (Listing 11).

Listing 9: **lsltempdatetime** imports.

```
28 import datetime
29 import time
```

Listing 10: **lsltempdatetime** constants.

```
31 SET_WINDOW_START_AND_END = -1
32 SET_WINDOW_END_AND_DURATION = 0
33 SET_WINDOW_START_AND_DURATION = 1
34
35 WINDOW_START_IS_BEGINNING_OF_TIME = -1
36 WINDOW_START_IS_OTHER = 0
37
38 WINDOW_END_IS_NOW = -1
39 WINDOW_END_IS_OTHER = 0
```

Listing 11: **lsltempdatetime** function.

```
42 def get_time_window(
43         window_range, start_marker, end_marker, start, end, duration):
44     """
45     Get a time window to be used in data display.
46 
47     Parameters
48     ----------
49     window_range : lsltempdatetime const
50         must be one of [SET_WINDOW_START_AND_END, SET_WINDOW_END_AND_DURATION,
51         SET_WINDOW_START_AND_DURATION].
52     start_marker : lsltempdatetime const
53          must be WINDOW_START_IS_BEGINNING_OF_TIME or WINDOW_START_IS_OTHER.
54     end_marker : lsltempdatetime const
55         must be WINDOW_END_IS_NOW or WINDOW_END_IS_OTHER.
56     start : datetime
57         Begining of data display window.
58     end : datetime
59         End of data display window.
60     duration : float
61         Length of datetime window in seconds.
62 
63     Raises
64     ------
65     ValueError
66         Range given is not one of the predefined constants.
67 
68     Returns
69     -------
70     window_start : datetime
71         Actual start of data display window.
72     window_end : datetime
73         Actual end of the data display window.
74 
75     """
76     if window_range == SET_WINDOW_START_AND_END:
77         window_start, window_end = _get_time_window_given_start_end(
78             start_marker, end_marker, start, end)
```

6.2.3 lsltempcontrols

The controls of the **lsltemp** GUI are defined in this module. To enable the generation of the controls various modules, that have been previously described, are imported into **lsltempcontrols** (Listing 12). One "public" class is defined in the module (Listing 13).

Listing 12: **lsltempcontrols** imports.

```
15 import wx
16 import libgsalt.gui.labelledcontrols as lc
17 import libgsalt.gui.icons as icons
18 import libgsalt.barebonesunits.barebonesunits as bbunits
19 import lsltempbase
20 import lsltempdatetime
```

Listing 13: **lsltempcontrols** classes.

```
26 class MainWindowControls(wx.Frame):
27     """
28     The class provides the main interface for the application.
29 
30     Extends wx.Frame.
31 
32     """
33 
34     def __init__(self, parent, _id, title):
```

6.2.4 lsltempdatatools

This module provides functions for reading data from probes using calls to modules that have already been described. To do this, **lsltempdatatools** imports the modules in Listing 14.

Listing 14: **lsltempdatatools** imports.

```
15 from pathlib import Path
16 import pandas as pd
17 import libgsalt.barebonesunits.barebonesunits as bbunits
18 import libgsalt.instrument.ithxtemperaturehumidityprobereader as ithxprobereader
19 import libgsalt.instrument.ithxtemperaturehumidityfilereader as ithxfilereader
20 import lsltempbase
```

Several "public" functions are defined (Listings 15, 16, 17, 18, 19).

Listing 15: **lsltempdatatools** functions (I).

```
31 def find_probes(callers_declared_probes):
32     """
33     Find all the probes that have been declared by the calling function.
34 
35     Parameters
36     ----------
37     callers_declared_probes : dict
38         {probe_id: {'name': probe_name, 'address_or_path': probe_address}}.
39 
40     Returns
41     -------
42     all_probes_declared : dict
43         Update to the caller defined probes to include simulated probes.
44     ids_of_working_probes : list
45         All the probes valid for use.
46     simulating_probes : bool
47         Return True if some probes are being simulated.
48 
49     """
50     simulating_probes = False
```

Listing 16: **lsltempdatatools** functions (II).

```
def get_probe_status(probe_address):
    """
    Get information about a probe's battery, recording status and clock.

    Parameters
    ----------
    probe_address : `str` : must be a valid IP address
        The address of the probe.

    Returns
    -------
    ithxprobereader dict
        {"ac_power_status": str, "battery_status": str,
         "recording_status": str,
         "computer_and_probe_clock_delta": str,
         "status_if_off_nominal": bool,
         "need_to_start_recording": bool}.

    """
    return ithxprobereader.get_probe_status(probe_address)
```

```
def get_sensor_memory_data_formatted_for_plot(
        probes, start_date, end_date, downloads_directory, averaging_frequency,
        temperature_units):
    """
    Read data from sensor memories and return it formatted for plotting.

    Parameters
    ----------
    probes : dict
        The probe identifiers in format {probe_id: {
            'name': name, 'address_or_path':address}}.
    start_date : datetime obj
        The start of the date range for data to be returned.
    end_date : datetime obj
        The end of the date range for the data to be returned.
    downloads_directory : `str` : must be valid path with write permission
        Folder to which data from probe will be written.
    averaging_frequency : libgsalt.instrument.ithxfilereader const
        Any averaging that will be applied to the data before display.
    temperature_units : `str` : must be libgsalt.barebonebunits units
        Units in which the data from probe will be fetched.

    Returns
    -------
    probes_with_data : dict
        The probes from the selection provided that have data.
    plot_dataframes : `list` : pandas dataframes
        The data obtained from the probes.

    """
    probes_with_data = {}
```

Listing 17: **lsltempdatatools** functions (III).

```
def get_file_data_formatted_for_plot(
        files, start_date, end_date, averaging_frequency, temperature_units):
    """
    Read the data in files and return it formatted for plotting.

    Parameters
    ----------
    files : dict
        File info in the format {file_id: {'name': str, 'address_or_path':path}}.
    start_date : datetime obj
        Start of the date range for the data to be fetched.
    end_date : datetime obj
        End of the date range for the data to be fetched.
    averaging_frequency : libgsalt.instrument.ithxfilereader const
        Any averaging to be applied to the data.
    temperature_units : `str` : must in libgsalt.barebonesunits units
        Units in which the data must be fetched.

    Returns
    -------
    files_with_data : dict
        Files that have data in the form {probe_id: probe_name}.
    file_dataframes : `list` : pandas dataframes
        The data obtained from the files.

    """
    files_with_data = {}
```

```
def combine_sensor_memory_and_file_data_for_plot(
        probes, probe_dataframes, files, file_dataframes):
    """
    Put probe and file data in the single dataframe.

    Parameters
    ----------
    probes : dict
        Probe identities in the format:
          {probe_id: {'name': name, 'address_or_path':path}}.
    probe_dataframes : `list` of pandas dataframes
        Datframes containing data from the probe memories.
    files : dict
        File identifiers in the format:
          {file_id: {'name': name, 'address_or_path':path}}.
    file_dataframes : `list` of pandas dataframes
        Dataframes containing data from the files.

    Returns
    -------
    probes_and_files : dict
        Identifiers for all the probes and files.
    probe_and_file_dataframe : `list` of pandas dataframes
        Data from the probes and files.

    """
    probes_and_files = {}
```

Listing 18: **lsltempdatatools** functions (IV).

```
247 def get_data_formatted_for_export_from_sensor_memory(
248         probe_id, probe_address, start_date, end_date, downloads_directory,
249         averaging_frequency, temperature_units):
250     """
251     Get data recorded in sensor memory and return it formatted for export.
252 
253     Parameters
254     ----------
255     probe_id : str
256         Identity of the probe.
257     probe_address : `str` : must be a valid IP address
258         Address of the probe.
259     start_date : datetime obj
260         Start of the date range for the data to be fetched.
261     end_date : datetime obj
262         End of the date range for the data to be fetched.
263     downloads_directory : `str` : must be a valid path with write permission
264         Directory where the data from sensor will be written before export.
265     averaging_frequency : libgsalt.instrument.ithxfilereader const
266         Any averaging that will be applied to the data.
267     temperature_units : `str` must be in libgsalt.barebonesunits units
268         The units for the temperature values exported.
269 
270     Returns
271     -------
272     data_found : bool
273         Return True if there is data in the date range specified.
274     dataframe : pandas dataframe
275         The data that has been found formatted for export.
276         Empty dataframe in no data in the specified date range.
277 
278     """
279     data_found, dataframe = _get_sensor_memory_data(
280         probe_id, probe_address, start_date, end_date, downloads_directory,
281         averaging_frequency, temperature_units)
```

Listing 19: **lsltempdatatools** functions (V).

```
def get_data_formatted_for_export_from_downloaded_sensor_file(
        file_id, file_path, start_date, end_date, averaging_frequency,
        temperature_units):
    """
    Get data downloaded from a sensor and return it formattted for export.

    Parameters
    ----------
    file_id : str
        The identity of the file from which data will be exported.
    file_path : `str` : must be path
        File for the data that is to be exported.
    start_date : datetime obj
        Start of the date range for the data that will be exported.
    end_date : datetime obj
        End of the date range for the data that will be exported.
    averaging_frequency : libgsalt.instrument.ithxfilereader const
        Any averaging that will be applied to the data.
    temperature_units : `str` must be in libgsalt.barebonesunits units
        The units for the temperature values exported.

    Returns
    -------
    bool
        Whether or not any data was found in the date range specified.
    pandas dataframe
        The data that was found. Empty dataframe if no data are found.

    """
    data_found, dataframe = _read_file_into_dataframe(
        file_id, file_path, start_date, end_date, averaging_frequency,
        temperature_units)
```

```
def check_averaging_consistent_with_display_period(
        display_window_start, display_window_end, averaging):
    """
    Consistency check for the averaging and display settings.

    Parameters
    ----------
    display_window_start : datetime obj
        Start of the data display window.
    display_window_end : datetime obj
        End of the data display window.
    averaging : libgsalt.instrument.ithxbase const
        The setting for averaging that will be used.

    Returns
    -------
    bool
        True if the averaging period is shorter than the display window.

    """
    return ithxfilereader.check_averaging_consistent_with_display_period(
        display_window_start, display_window_end, averaging)
```

6.2.5 lsltempplots

The final module, to be described, provides plotting tools. Several modules already described are also imported into this module (Listing 20). One "public" function, Listing 21 for obtaining probe names and addresses is defined. Two classes are defined for obtaining live streamed data (including from `Simulated` instruments) as well as data from sensor memory and files (Listing 22).

Listing 20: **lsltempplots** imports.

```
25 import datetime
26 import numpy as np
27 from matplotlib.lines import Line2D
28 from matplotlib import pyplot as plt
29 import matplotlib.dates as mdates
30 import matplotlib.animation as animation
31 import lsltempbase
32 import libgsalt.barebonesunits.barebonesunits as bbunits
33 import libgsalt.instrument.ithxtemperaturehumidityprobereader as ithxprobereader
```

Listing 21: **lsltempplots** functions.

```
50 def get_probe_names_and_addresses(probes):
51     """
52     Get names, addresses and a count of probes from the probes definition.
53 
54     Parameters
55     ----------
56     probes : dict
57         The descriptors of the probes that have been declared.
58 
59     Returns
60     -------
61     probe_ids : list
62         The IDs of the probes that have been found.
63     probe_names : list
64         The names of the probes that have been found.
65     number_of_probes : int
66         The number of probes that have been found.
67 
68     """
69     probe_ids = []
```

Listing 22: **lsltempplots** classes.

```
223 class LiveSensorDataDisplayScope(_DataDisplayScope):
224     """
225     The class provide a display for live data plotting.
226 
227     Extends _DataDisplayScope
228     """
229 
230     def __init__(self, probes, update_period, plot_series, plot_time_window,
231                  temperature_units):
```

```
319 class SensorMemoryOrFileDataDisplayScope(_DataDisplayScope):
320     """
321     The class provide a display for sensor memory or file data plotting.
322 
323     Extends _DataDisplayScope.
324 
325     """
326 
327     DATE_TIME_FORMAT = "%Y/%m/%d␣%H:%M:%S"
328 
329     def __init__(self, probes_and_files, dataframe, plot_series,
330                  plot_time_window, temperature_units):
```

6.3 The configuration file

The probes that the application uses and their addresses are defined in a **JSON** configuration file such as shown in Listing 23. Upon startup, the application reads this file and attempts to communicate with the probes and determine their status (Section 3.3).

Listing 23: **lsltemp** settings.

```
{   "recording_interval_in_seconds": "60",
    "probes": {
        "LAB-ONE":
            {
                "name": "Lab 1",
                "sensor": "Omega iTHX-SD",
                "address_or_path": "XXX.XXX.XXX.XXX"
            },
        "LAB-TWO":
            {
                "name": "Lab 2",
                "sensor": "Omega iTHX-SD",
                "address_or_path": "XXX.XXX.XXX.XXX"
            },
        "LAB-THREE":
            {
                "name": "Lab 3",
                "sensor": "Omega iTHX-SD",
                "address_or_path": "XXX.XXX.XXX.XXX"
            }
    }
}
```

6.4 Generating Microsoft Windows installers

In addition to the Python code described above and which can be run in a suitable environment, preferably a virtual Python environment such as those that can be set up with venv[6] or Anaconda[7], **lsltemp** has been successfully packaged as a Microsoft Windows installer. Thus it can be installed and run in Windows 10 (tested) or possibly other versions (≥Windows 7, but not tested) without the need to access any of the underlying code. Compilation of the Windows installer was done using PyInstaller[8]. To build the windows installer, on a Debian Linux platform (https://www.debian.org/), Python was installed on top of a Wine compatibility layer (https://www.winehq.org/) capable of running Windows applications on several POSIX-compliant operating systems. Some screenshots obtained when running the Windows version of the application are shown in Appendix A.

References

1. *Python 3.8.2 documentation*. https://docs.python.org/3/index.html. URL: https://docs.python.org/3/index.html (visited on 05/17/2020).
2. N. Rappin and R. Dunn. *WxPython in Action*. Manning Publications, 2006. ISBN: 9781932394627. URL: https://books.google.com/books?id=RgvFQgAACAAJ (visited on 05/17/2020).
3. Omega Engineering. *User's Guide: iServer MicroServer, iTHX-SD Temperature + Humidity*. Omega Engineering, Inc., 2013.
4. Wes McKinney. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc., 2017. ISBN: 978-1-4919-5766-0.
5. Wes McKinney and the Pandas Development Team. *pandas: powerful Python data analysis toolkit Release 1.0.3*. 2020. URL: https://pandas.pydata.org/pandas-docs/stable/pandas.pdf (visited on 05/17/2020).
6. *venv - Creation of virtual environments*. URL: https://docs.python.org/3/library/venv.html (visited on 07/18/2020).
7. *Anaconda Software Distribution*. URL: https://anaconda.com/ (visited on 07/18/2020).
8. David Cortesi. *PyInstaller Documentation: Release 3.6*. 2020. URL: https://readthedocs.org/projects/pyinstaller/downloads/pdf/stable/ (visited on 05/17/2020).

Appendix A

Microsoft Windows Version of the Application

The **lsltemp** application comes as multi-platform Python source code as well as a Microsoft (MS) Windows installer. Here, we show some screenshots from the MS Windows version running on Windows 10. Figure A1 shows the graphical user interface of the application on startup. Figures A2 and A3 show file selection and data visualization respectively. Figure A4 shows examples of some of the application's modal dialogs as they appear on Windows 10.

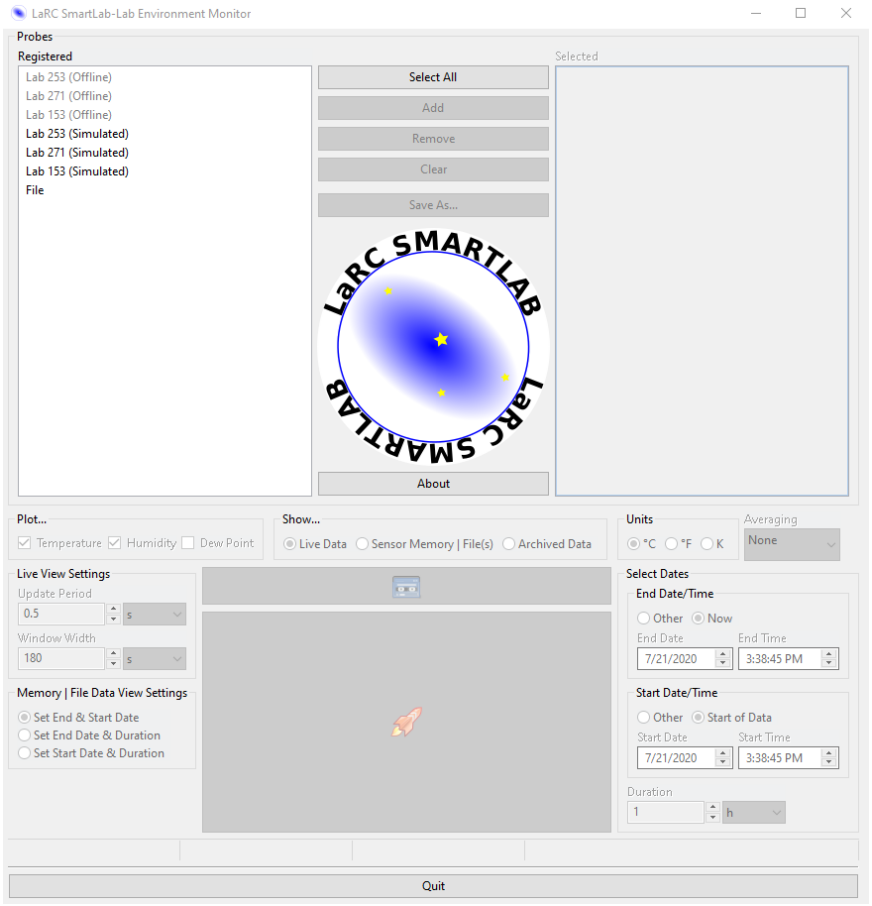


Figure A1: The application GUI on startup on MS Windows 10.

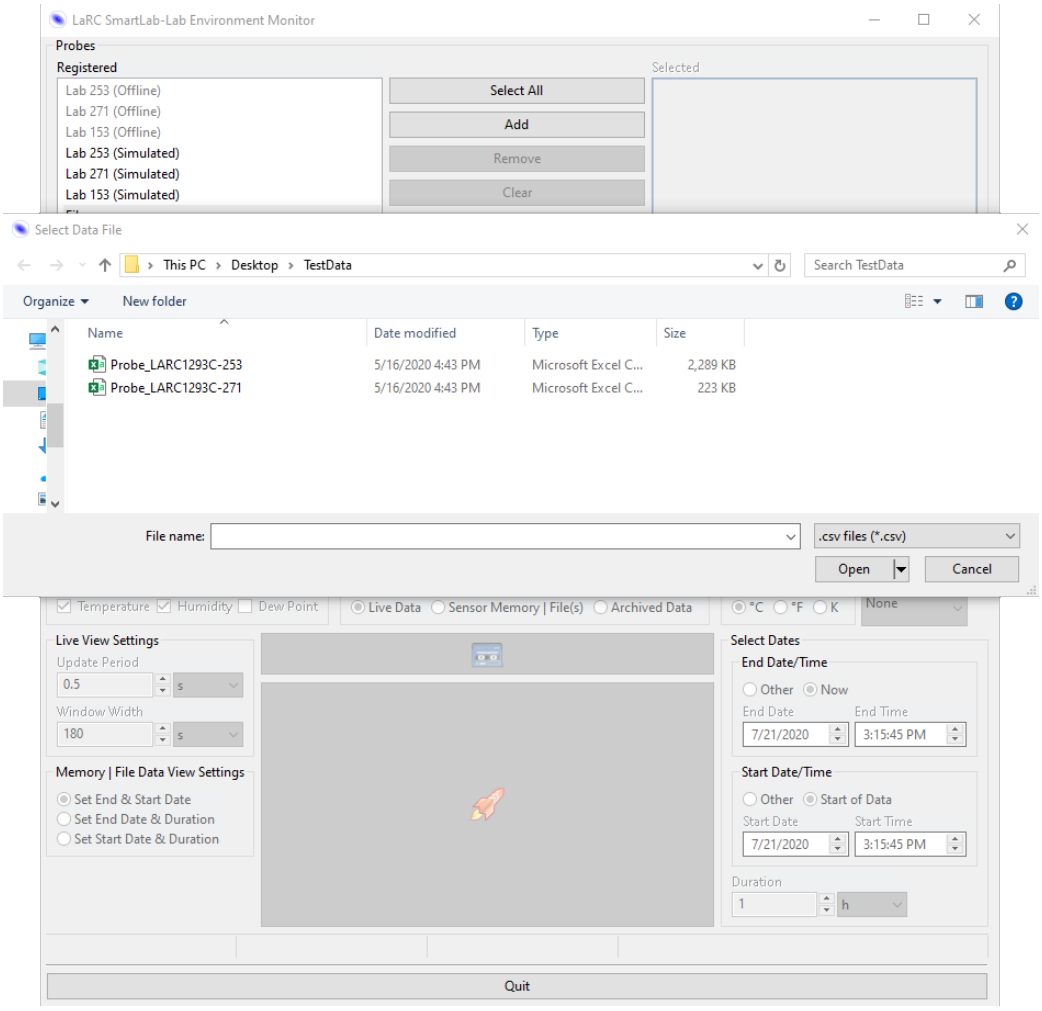
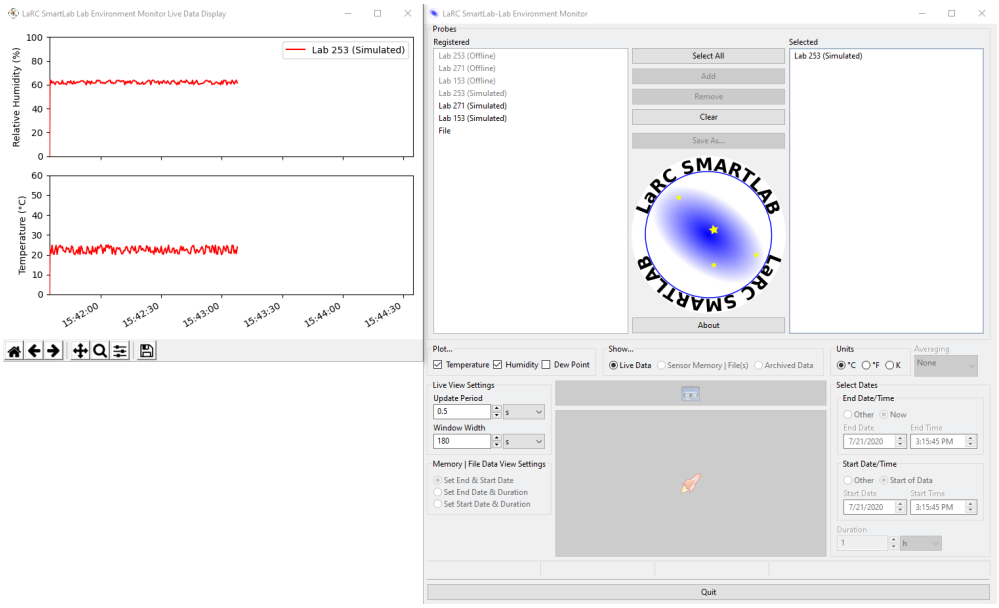
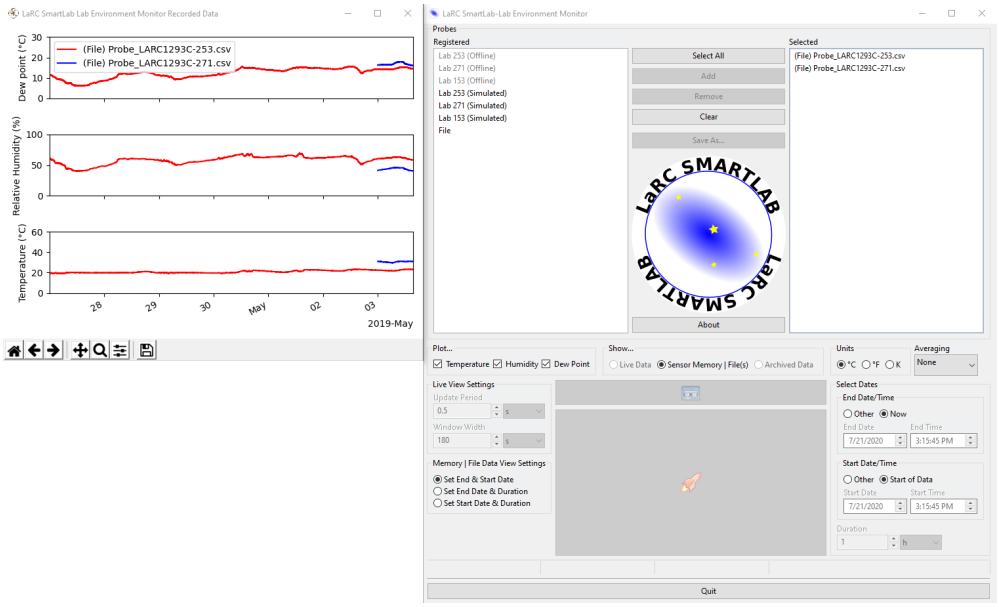


Figure A2: File selection dialog box on MS Windows 10.

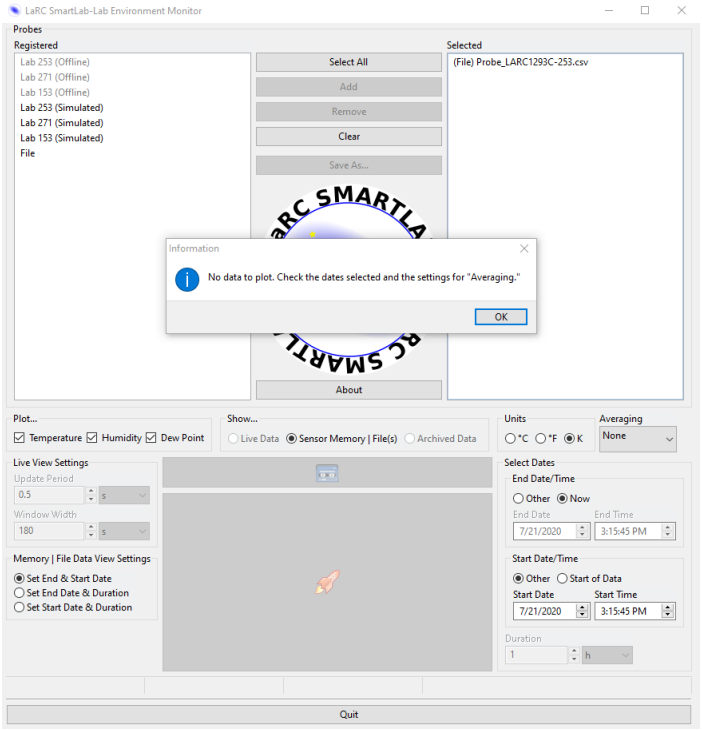


(a) "Live" data from a simulated probe

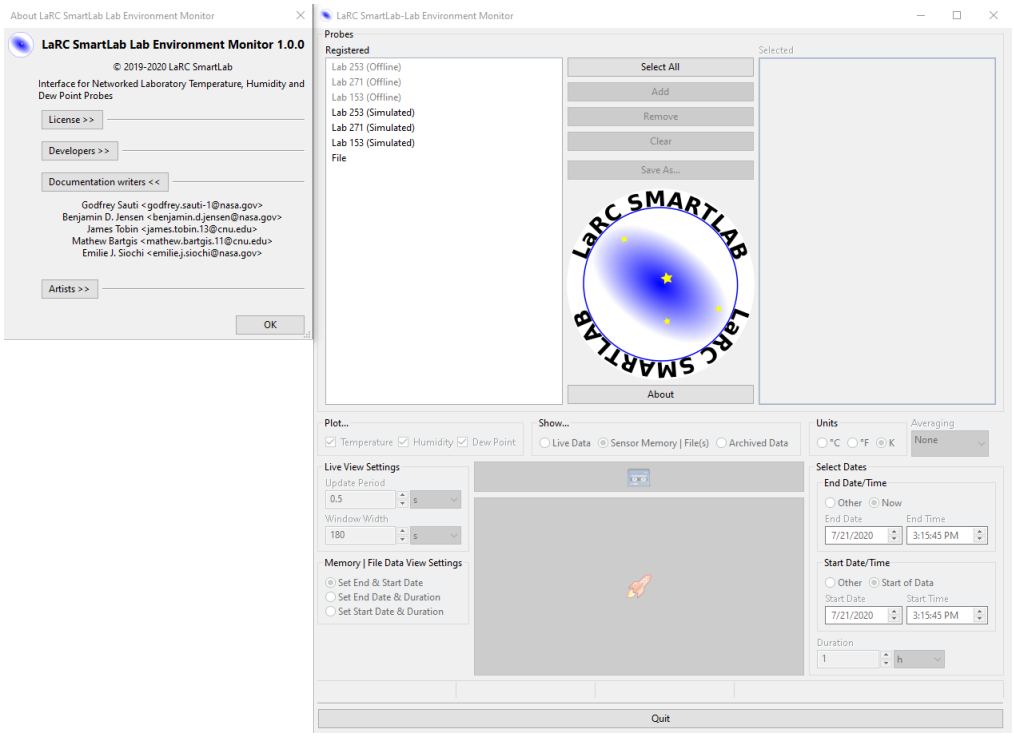


(b) Displaying previously downloaded data from files

Figure A3: Displaying live sensor data as well as data from files on MS Windows 10.



(a) Invalid date range/data averaging information



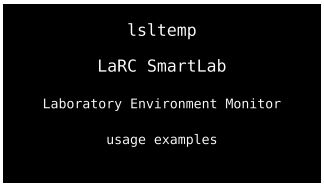
(b) About

Figure A4: Examples of the modal dialogs displayed on MS Windows 10.

Appendix B

Video Examples

The following videos showing use of the application are available as supplementary information:



(a) Introduction (All)

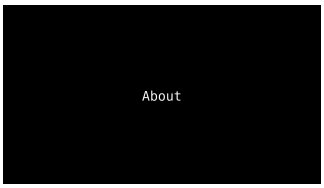
(b) Video 1

(c) Video 2

(d) Video 3

(e) Video 4

(f) Video 5



(g) About (All)

Figure B1: Section headers from the videos with examples of the usage of the application.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704–0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-08-2020	Technical Memorandum	

4. TITLE AND SUBTITLE
LaRC SmartLab Apps For Instrument Control and Data Processing: Laboratory Environment Monitor

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Godfrey Sauti, Benjamin D. Jensen, James D. Tobin, Mathew L. Bartgis, Emilie J. Siochi

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
736466.01.08.07.20.59.02

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, Virginia 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)
NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NASA/TM-2020-5006086

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Category 61
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES
An electronic version can be found at http://ntrs.nasa.gov.

14. ABSTRACT

The **LaRC SmartLab** applications are a series of software tools to greatly enhance researcher efficiency by streamlining and automating workflows. Python scripts and applications are increasingly being used in scientific workflows, including for instrument control and data processing. Interactive Python scripting environments such as JupyterLab provide powerful tools for using Python. In some use cases, the development of standalone applications with dedicated graphical user interfaces can enhance the utility of the code and open it up to more users, including non-programmers. Here, we describe a Python based application for communicating with, and displaying data from, iTHX Temperature, Humidity, and Dew Point probes. We discuss the set up and use of the application as well as its implementation. We also highlight the use of `Simulated` probes to enable users and developers to familiarize with or debug the application, even when they do not have access to the physical hardware in the laboratory.

15. SUBJECT TERMS
LaRC SmartLab, Python Programming, Graphical User Interfaces (GUIs), wxPython, `Simulated` Probes, iTHX Probes, Laboratory Environment Monitoring, Temperature, Humidity, Dew Point, Debian Linux, Internet of Things (IOT)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Information Desk (`help@sti.nasa.gov`)
U	U	U	UU	65	**19b. TELEPHONE NUMBER** *(Include area code)* (757) 864-9658